

VÁCLAV BELÁK

ONTOLOGY-DRIVEN SELF-ORGANIZATION OF
POLITICALLY ENGAGED SOCIAL GROUPS

<http://www.ontopolis.net>

UNIVERSITY OF ECONOMICS, PRAGUE
FACULTY OF INFORMATICS AND STATISTICS
DEPARTMENT OF INFORMATION TECHNOLOGY



Study Programme: Applied Informatics

Specialization: Cognitive Informatics

ONTOLOGY-DRIVEN SELF-ORGANIZATION
OF POLITICALLY ENGAGED SOCIAL
GROUPS

VÁCLAV BELÁK

Master's Thesis

Supervisor: Doc. Ing. Vojtěch Svátek, Dr.

Oponent: Mgr. Vít Nováček

Prague, 2010

DECLARATION

I hereby declare that I elaborated this thesis on my own and that all literature and other sources are correctly mentioned in it.

Prague, 2010

Václav Belák

This thesis is dedicated to my parents.

ABSTRACT

This thesis deals with the use of knowledge technologies in support of self-organization of people with joint political goals. It first provides a theoretical background for a development of a social-semantic system intended to support self-organization and then it applies this background in the development of a core ontology and algorithms for support of self-organization of people. It also presents a design and implementation of a proof-of-concept social-semantic web application that has been built to test our research. The application stores all data in an RDF store and represents them using the core ontology. Descriptions of content are disambiguated using the WordNet thesaurus. Emerging politically engaged groups can establish themselves into local political initiatives, NGOs, or even new political parties. Therefore, the system may help people easily participate on solutions of issues which are influencing them.

Keywords: ontology, self-organization, politics, WordNet, the semantic web, RDF, OWL, e-democracy

ABSTRAKT

Tato práce se zabývá využitím znalostních technologií pro podporu samoorganizace lidí sdílejících společné politické cíle. V první části nejprve teoreticky zakládá možnost tvorby sociálně-sémantického systému pro podporu samoorganizace a v druhé části tyto poznatky aplikuje při vytvoření nové jádrové ontologie, algoritmů pro podporu samoorganizace a vytvoření testovací sociálně-sémantické aplikace využívající jakožto datový model RDF a disambiguující popis těchto dat pomocí tezauru WordNet. Emergující politicky angažované skupiny občanů se pak mohou etablovat v lokální politické iniciativy, nevládní organizace či dokonce v nové politické strany. Systém tak umožňuje lidem se spolupodílet na tvorbě řešení problémů, které je obklopují.

Klíčová slova: ontologie, samoorganizace, politika, WordNet, sémantický web, RDF, OWL, e-demokracie

PUBLICATIONS

Some ideas and figures will appear in the following publication:

- Václav Belák and Vojtěch Svátek. Ontopolis.net: Social-Semantic Web Application for Participative e-Democracy. In *Proceedings of the 9th Znalosti 2010 Conference*. 2010 (to appear).

Interesting phenomena occur when two or more rhythmic patterns are combined, and these phenomena illustrate very aptly the enrichment of information that occurs when one description is combined with another.

— Gregory Bateson

ACKNOWLEDGMENTS

I would like to thank a lot my parents, who always were with me still ready to support, even if they pretty knew I was wrong. This way I would also like to thank my lovely Zuzana, who has kindly slept-through all my quite coding and thinking as well as always supported me in what I was doing.

I would like to thank various members of Knowledge Engineering Group¹ from the Department of Information and Knowledge Engineering² for their support. RNDr. Radomír Palovský kindly provided me with a testing server and together with Ing. Tomáš Kliegr helped me to get it work. Bc. Jan Zemánek gave me a plenty of his time deep-diving into Jena and SPARQL and last but not least the most important was the help, friendly guidance and critical but yet open-minded advices of Doc. Vojtěch Svátek.

I also thank to all my friends and colleagues, who criticized the ideas of this thesis and thus helped to uncover some of its mistakes. Especially the interviews with Ing. Jan Burian, Dr. Cyril Brom and Mgr. Jiří Fiala helped to shape the essential ideas of the thesis.

¹ See <http://keg.vse.cz>.

² See <http://kizi.vse.cz>.

CONTENTS

| | |
|--|-----------|
| I THEORETICAL BACKGROUND | 1 |
| 1 INTRODUCTION | 3 |
| 1.1 Motivation | 3 |
| 1.2 Main objectives | 3 |
| 1.3 Related work | 4 |
| 2 SELF-ORGANIZATION IN POLITICS | 7 |
| 2.1 Complex systems | 7 |
| 2.2 Self-organization | 8 |
| 2.3 Complex networks | 9 |
| 2.4 Collective intelligence and politics | 10 |
| 2.5 Implications on the system design | 12 |
| 3 KNOWLEDGE TECHNOLOGIES | 13 |
| 3.1 The Semantic Web | 13 |
| 3.1.1 RDF | 14 |
| 3.1.2 OWL | 16 |
| 3.1.3 SPARQL | 18 |
| 3.1.4 SPARUL | 19 |
| 3.2 WordNet thesaurus | 19 |
| 3.3 Semantic similarity | 20 |
| II ONTOPOLIS PROJECT ACHIEVEMENTS | 23 |
| 4 DEVELOPED ONTOLOGY | 25 |
| 4.1 Incorporating DCTerms | 25 |
| 4.2 Incorporating FOAF | 26 |
| 4.3 Incorporating SIOC | 27 |
| 4.4 Incorporating DOLCE for representation of political programs | 29 |
| 4.4.1 Representing political programs | 30 |
| 4.5 Representation of word sense disambiguation | 32 |
| 4.6 Representing trust: incorporating Konfidi | 34 |
| 5 ONTOPOLIS.NET | 37 |
| 5.1 Architectural constraints and requirements | 37 |
| 5.2 Architecture and implementation of Ontopolis.net | 38 |
| 5.2.1 Grails framework | 39 |
| 5.2.2 Use of Jena for working with RDF | 41 |
| 5.2.3 Use of Pellet for constraint validation | 45 |
| 5.2.4 Programming with ontology | 46 |
| 5.2.5 Similarity algorithms | 47 |
| 5.2.6 User interface | 49 |
| 6 CONCLUSION AND FUTURE WORK | 51 |
| BIBLIOGRAPHY | 55 |
| III APPENDIX | 61 |
| A ONTOPOLIS SCHEMA | 63 |
| B SIMILARITIES BETWEEN SOLUTION PLANS | 73 |
| C SPARQL SIMILARITY QUERIES | 77 |

LIST OF FIGURES

| | | |
|-----------|--|----|
| Figure 1 | Scale-free complex network | 10 |
| Figure 2 | Example of RDF in Turtle | 14 |
| Figure 3 | Diagram of RDF reification | 15 |
| Figure 4 | RDF reification in Turtle | 15 |
| Figure 5 | Simple SPARQL statement | 18 |
| Figure 6 | Simple SPARUL statement | 19 |
| Figure 7 | Fragment of WordNet | 22 |
| Figure 8 | FOAF vocabulary used in OPOL | 27 |
| Figure 9 | Overview of SIOC core ontology (source: [SIOCC]) | 28 |
| Figure 10 | SIOC vocabulary used in OPOL | 28 |
| Figure 11 | Dependencies between used DOLCE modules | 30 |
| Figure 12 | Elements of DOLCE used in OPOL | 31 |
| Figure 13 | Example of solution plan | 31 |
| Figure 14 | WordNet Basic schema and its use in disambiguation | 33 |
| Figure 15 | Similarity relations representation | 34 |
| Figure 16 | Trust representation in OPOL | 36 |
| Figure 17 | General architecture of Ontopolis.net | 39 |
| Figure 18 | Closure in Groovy | 40 |
| Figure 19 | Transaction handling example | 42 |
| Figure 20 | Connection management in Jena | 44 |
| Figure 21 | Rule for relating groups to persons | 45 |
| Figure 22 | The layout of Ontopolis.net | 49 |

LIST OF TABLES

| | | |
|---------|---|----|
| Table 1 | Dublin Core metadata terms used in OPOL | 26 |
|---------|---|----|

ACRONYMS

| | |
|------|-----------------------------------|
| NGO | Nongovernmental Organization |
| API | Application Programming Interface |
| HTTP | HyperText Transfer Protocol |
| www | World Wide Web |
| HTML | HyperText Markup Language |
| CSS | Cascading Style Sheets |

RDF Resource Description Framework
URI Uniform Resource Identifier
XSD XML Schema Definition
XML eXtensible Markup Language
OWL Web Ontology Language
RDFS RDF Schema
SPARQL SPARQL Protocol and RDF Query Language
SQL Structured Query Language
SPARUL SPARQL/Update Language
DML Data Manipulation Language
FOAF Friend Of A Friend
N₃ Notation 3
Turtle Terse RDF Triple Language
SIOC Semantically-Interlinked Online Communities
DCMI Dublin Core Metadata Initiative
DOLCE Descriptive Ontology for Linguistic and Cognitive Engineering
DnS Descriptions and Situations
OOP Object Oriented Programming
CWA Closed World Assumption
OWA Open World Assumption
UNA Unique Names Assumption
GSP Groovy Server Pages
MVC Model View Controller
POGO Plain Old Groovy Object

Part I

THEORETICAL BACKGROUND

INTRODUCTION

Democracy is the worst form of government, except for all those other forms that have been tried from time to time.

— Winston Churchill

1.1 MOTIVATION

Political systems in present democratic countries are more-or-less built on top of competition of several political parties. This competition is at least theoretically a source of new ideas and better services for the electorate. However, it is often biased by personal relationships between secretaries of political parties and elite politicians, because secretaries have the power to decide the order of candidates on the candidate list and thus influence the ability of a particular candidate to succeed. Although in the Czech republic there exists an option to choose at most two preferred candidates on the list during the parliamentary poll and thus influence the order, this option is generally not used very often. For instance, in the last parliamentary poll in the Czech republic in 2006, this preferential votes enable only 6 candidates at all to succeed.¹[CSB]

The competition between political parties of our time resembles oligopoly from economic theory, where the market is managed only by a few subjects which control price and trend of the whole market branch. The competition is thus rather a source of continuous verbal struggle than a generator of new ideas and solutions, which in turn discourages potential new candidates from participation in political life. This situation leads, beside other effects, to a big gap between politics and civil society in the Czech republic.

Recent widespread use of so-called social-web or web 2.0² applications points to the new possibilities of decentralized large-scale collaboration and self-organization which have been unthinkable in the pre-Internet era. The aim of the thesis is to combine this possibilities with the knowledge technologies of the semantic web to help citizens self-organize and collaboratively solve public issues.

1.2 MAIN OBJECTIVES

General vision of the system described in this thesis is a platform for building communities around various topics in politics. It aims to be something like modern and (hopefully) more intelligent way of ancient Greek's polis, where free citizens express their opinions and suggest solutions, but in a massively distributed and decentralized way that is allowed by the Internet and the state-of-the-art artificial intelligence technology. We have lost this possibility to a great extent

¹ The Czech Chamber of Deputies has 200 members.

² Note that Web 2.0 is based on the exact same standards and technologies as the "classic" web. On the contrary, the semantic web technologies extend this "classic" web to describe the meaning of information and thus allow the machine processing of content on the web. The semantic web is a topic of the section [3.1](#).

because our societies are much more bigger than those of ancient Greek. Actually, that was one of the reason why ancient Greeks asked their youth people to create new towns when their hometown had become a metropolis.[Müller, 2008, p. 63]

Social web applications is today's hot topic. Facebook.com, LinkedIn.com or Wikipedia.org are very popular among the users of the web. What these services have in common is that they allow users to connect to their friends and/or to do some activities (e.g. to create content) jointly. Standards and technologies that these services are built on do not allow to express the meaning of the information or they allow it in a really constrained and implicit way. So that it is very hard to process this information in an effective and intelligent manner on computers. The Semantic Web initiative³ addresses this constraint by developing new standards and technologies which allow to express meaning explicitly while turning the web into the huge distributed knowledge base. Presented system is a social-semantic web application, because it allows people to collaboratively solve political problems and helps them self-organize together and to do this precisely it needs to consider the actual meaning of the information, therefore it is built on top of knowledge standards and technologies like RDF, OWL and Jena.

The main goal of building the system consists of four sub-goals. First is to develop a core ontology for description of political programs, commitments and trust between people, as well as their mutual relationships. Second is to develop new and accommodate existing algorithms for support of self-organization of people. Namely, algorithms for computing of semantic similarity between program descriptions and for enhanced social recommendation were researched. Third is to develop a prototype of the social-semantic web application to test the ontology and the algorithms. The last sub-goal is to contribute to the discussion of the role of information and knowledge technologies in e-government and e-democracy in particular and the role of a citizen in democracy in general.

Section 3.1 on page 13 discuss these technologies in more detail.

1.3 RELATED WORK

In present, several similar web applications either exist or are actively developed. Openpolitics.ca is focused on similar topics but uses different approach. It is a wiki-based system where issues are organized by naming convention, i.e. it does not use any formal ontology language. There is no support for social network self-organization nor issue matching. Localocracy.org is described to be a social application to help people collaboratively *"identify the most critical needs of their community and debate and popularize innovative and efficient ways of meeting them"*. [LoCY] This site is being developed by the start-up company and there is a little information available on the web, but according to their description the general idea is very similar to ours. Whitehouse2.gov is also a web application where every user can declare his/her priorities in the political world or vote on the priorities of others. General overview is then presented on the main page of the site. Zmenpolitiku.cz is another web application very similar to Whitehouse2.gov. Users are divided between two groups. The first group consists of what could

³ See <http://www.w3.org/2001/sw/>.

be called opinion leaders, because their opinions appear on the top of the main page of the site. A professional politician can not be a member of this group and the members were selected according to public opinion poll realized via Facebook.com. The other group consists of the rest of people who joined the site. This site provides opportunities to organize electronic petitions and declare (dis)agreement with opinion of someone else. Apparently neither of these two, i.e. Zmenpolitiku.cz nor Whitehouse2.gov, is using any formal ontology and thus it is hard to get integrated overview about citizen opinions. In addition, these sites are more like a political barometer than a collaborative platform for emergent political action. Van Atteveldt [2008] describes various ontologies for description of political reality and the approach chosen in his work for formalizing political roles and issues has been our source of inspiration in respective parts of the thesis that are described in full details in the chapter 4 on page 25.

The remainder of the thesis is organized into two parts. The first part provides a theoretical background in self-organization, collective intelligence and knowledge technologies. In the second part we apply these theories and technologies first in the creation of the core ontology for description of the political domain and then in the design and implementation of the proof-of-concept web application. The ontology re-uses several other schemas and thus we discuss these ontologies as well. The final chapter then concludes the thesis and presents intended extensions of the project in the future.

*Laws and institutions must go hand in hand with the progress
of the human mind . . . as new discoveries are made . . .
institutions must advance also, and keep pace with the times.*

— Thomas Jefferson

Many systems around us are *complex*. Our body, immune system, food chains, etc., are all examples of a complex system. Moreover, many of social systems like global markets, language, WWW and so on, are complex too. It is not our aim here to define a complexity, but it is rather to provide an intuitive insight into features characterizing complex systems. Especially social systems with enhanced capability of decision-making are of our particular interest in this chapter.

2.1 COMPLEX SYSTEMS

Every system consists of parts. In a complex system, these parts interact, so that they “*are both distinct and connected, both autonomous and to some degree mutually dependent*”. [Heylighen, 2008] In social systems, these interacting parts are e.g. persons, groups, institutions (*agents*, generally), that act in order to attain their aims. Each agent can have different ones. Sometimes they are antagonistic but sometimes they are not. In politics, for example, there are usually parties, which share a great deal of goals and thus they are able to collaborate in a coalition. On the other hand, there are also parties whose aims are mostly counter those of others (e.g. non-democratic parties). Interaction of agents in a complex system leads to processes which are often non-linear, because they are influenced by numerous *positive* and *negative* *feedbacks*. This causes unpredictability of future states, because such a system is very sensitive to initial conditions — this is sometimes called the *butterfly effect*. Consequently, complex systems are neither regular nor predictable. In contrast with e.g. a clock, which is both predictable and stable, a complex system is hard to predict, because it is continuously changing and because positive feedbacks allow even a small difference in a parameter to make a significant change in future states. For example, a social network is constantly changing because of new people and new relationships are continuously added into it, or older ones cease to exist. But people are usually networking via their actual friends. Therefore, if an initial structure of the social network is slightly different, it may lead to a very different structure in the future. Although it is unpredictable and irregular, complex system is also neither chaotic nor random. “*This intermediate position, balancing between rigidity and turbulence, is sometimes called the ‘edge of chaos’.*” (*ibid*) The reason why complex systems are both unpredictable but yet quite stable are their ability to *self-organize*.

2.2 SELF-ORGANIZATION

“Self-organization can be defined as the spontaneous emergence of global structure out of local interactions.”(ibid) In context of this definition the goal of this thesis, i.e. to create an ontology for *driving* a self-organization, may sound contradictory, because spontaneity of self-organization means that no one manages it. The main objective of this section is to clarify this conceivable contradiction and by doing this we also describe the concept of self-organization.

Gershenson and Heylighen [2003] analyzed conditions under which it is suitable to call a system self-organizing or even create an artificial one. One of their crucial findings is the fact that “organization is more than low entropy: it is structure that has a function or purpose” and this purpose “is not an objective property of the system, but something set by an observer”, so they conclude that “self-organization is a way of modelling systems, not a class of systems”.(ibid) If we choose a very short time interval, the global market will not be self-organizing. Likewise if we choose only a tiny fraction of it. But if we choose longer timescale, we perceive a whole symphony of billions of different goals orchestrated by the “invisible hand of market”.¹ Parts of the self-organizing system interact only locally at the beginning, so that distant parts are mutually independent. But as the organization structure of the system emerges in time, these distant parts begin to be dependent. This is caused by the fact that agents *prefer* only certain situations. For example, in an economy where there is no division of labour, there is no need of money, because everyone can do everything on its own. Therefore, the structure of economic relations in that hypothetical economy is very shallow. But as the division of labour diversifies an ability of agents to produce goods, it is much more rational to introduce some form of market to mutually enhance utility of each agent. It means that agents *prefer* the exchange of goods to the situation without market because of higher utility. Interactions of agents in the economy thus become *synergistic*. However, they have lost part of their initial independence. The whole system is now much more interdependent and can be perceived as an entity on its own, whose goal is to maximize the overall utility rather than the individual ones. New features of this entity, which is not a merely sum of its parts, are called *emergent*. For instance, a price in an economy that enables market clearing is such an emergent property. Despite of the fact a self-organizing system is intrinsically stable, it is not rigid. It *adapts* to outside perturbations to a great extent. The change in an amount of goods demanded leads to an increase in its price while the whole economy keeps going, for example.

The abovementioned conceivable contradiction is apparently ungrounded, because the view of observer defining the purpose of the system is arbitrary, and thus if we take into account the fact that the ontology is only a part of the whole system consisting of the ontology, the application and its users, there is no obstacle to conceiving the whole system as self-organizing.

¹ This mechanism of turning various aims of agents into a synergistic collaboration has also its significant pitfalls and limits. It is not our objective in this thesis to analyze them, however.

2.3 COMPLEX NETWORKS

Many complex systems can be often represented as a graph consisting of nodes and edges. In a social network, for instance, the nodes are people and the edges are their relationships. Typical feature of all complex networks is that they are networks of a *small-world*. This means that a path connecting two arbitrary distinct nodes is short with respect to the size of the network. Milgram's experiments in USA concludes that an average distance between two randomly selected persons is 5.5. This finding is sometimes referred to as *the six degrees of separation*. [Barabási, 2005, p. 33]

Another feature of complex networks is clustering. For example, in a social network if person X knows person Y and Y knows person Z, then it is very probable that X also knows Z. We can call these clusters of friends an *ego*.² The ego is linked to another one by a weak tie. When a person needs to solve a problem, e.g. to find a job, these weak ties become very important, because they connect him/her with people *outside* of his/her ego. That is to say, people inside the ego will probably not be very helpful, because they have very similar information. (*ibid*, p. 47) Typical complex network consists of highly connected hubs that are linked together by significantly less amount of links, because their link distribution follows a *power law*. More formally, "the number of nodes N with a given degree (i.e. number of links) K is proportional to a (negative) power of that degree" [Heylighen, 2008]:

$$N(K) \sim K^{-\alpha},$$

where α is typically within the interval $[1, 3]$. Networks whose link distribution follows this rule are called *scale-free*. The average distance between two randomly chosen nodes is small, because hubs serve as *shortcuts*.

Barabási [2005, p. 90] developed an iterative algorithm for generation of scale-free networks. The algorithm is defined by the following two rules:

1. *Growth*: A new node is added in each iteration.
2. *Preferential attachment*: Each new node connects to the two existing ones. The probability of choice of the node is directly proportional to the number of links (its degree) it already has.

The figure 1 depicts a social network consisting of twenty users that was generated by this algorithm.³ Number of the node depicts a step in which it was added to the network, beginning with 0. Notice that the earlier nodes are more connected to others than the older ones.

Given these two rules and supposing the social network of politically engaged people will grow, following question arises: *How can be the general preference of people in politics defined?* We believe the most important for people in politics is **trust**. Therefore, to support the process of self-organization of people in politics, it is our goal to provide a person with recommendations of the others, who are trustworthy with respect to his/her interests and goals.

² This term emphasizes the fact that an individual is surrounded by its similar-minded friends.

³ This network is a part of the testing data-set mentioned in the section 5 on page 37.

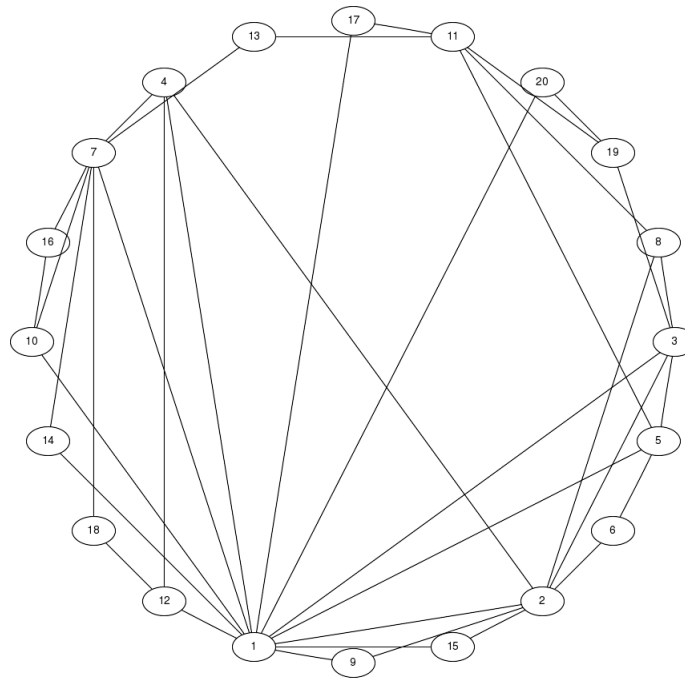


Figure 1: Scale-free complex network

2.4 COLLECTIVE INTELLIGENCE AND POLITICS

A self-organizing social system has sometimes emergent properties which allows it to solve problems that any individual it is consisted of can not solve on its own. An interdisciplinary team of scientists or a collective of users in a prediction market⁴ are examples. This phenomenon is called *collective intelligence*. [Heylighen, 1999] Another definition of collective intelligence by Malone et al. [2009] is more broad: “groups of individuals doing things collectively that seem intelligent”. To reject a conceivable misunderstanding of this term in the sense of emphasizing a role of a collective rather than an individual, Lévy [1999, p. 13] defines collective intelligence as “a form of universally distributed intelligence, constantly enhanced, coordinated in real time, and resulting in the effective mobilization of skills. . . . The basis and goal of collective intelligence is the mutual recognition and enrichment of individuals rather than the cult of fetishized or hypostatized communities.”

Agents need an environment to self-organize and to collaboratively and intelligently solve complex tasks. In context of collective intelligence, Heylighen [1999] proposes the environment with shared read/write access called *collective mental map*. To construct this environment, he specifies following mechanism:

1. averaging of individual contributions
2. amplifying beneficial signals via positive feedback loops
3. division of labour between agents with different domain of expertise

⁴ Prediction market is a system where people buy and sell possible answers to some question (e.g. “When will the war in Afghanistan end?”). The probability that the answer is right is then evaluated by its price. See [Rodriguez and Watkins, 2009].

Collective intelligence⁵ is a natural principle of collective decision making. A democratic parliament during voting can be an example. It consists of members with different domain of expertise and individual contributions are averaged via voting. According to [Rodriguez and Watkins \[2009\]](#), theoretical analysis of collective intelligence in decision making has its roots in The Age of Enlightenment. They mention the jury theorem of Marquis the Condorcet and summarize it as follows: “when a group of ‘enlightened’ decision makers chooses between two options under a majority rule, then as the size of the decision making population tends toward infinity, it becomes a certainty that the best choice is rendered”. They also point out to the fact that the result depends on the fact that voters are *enlightened*, so that in average they choose the best option. Because of this, prominent personalities of the Enlightenment suggested free and universal education. On the other hand, if the tendency is opposite, the majority rules voting ends up with the worst option chosen. This is the point where the third aforementioned condition applies. Voters have to argue and negotiate in order to allow better proposals to become popular among other members. However, reality in present parliaments is sometimes far away from ideals of the Enlightenment.

Our form of economic and political organization is to a great extent determined by communication infrastructure we use. The invention of democracy in ancient Greece is thus tightly connected with an emergence of alphabet, because it allowed more people to participate in the legislative process. Similarly, printing brought the possibility to print newspapers that are shaping public opinion and without which modern democracy is almost unthinkable. [[Lévy, 1999](#), p. 58] Advances in logistics, transportation and information technology have made our world globalized. Many of political problems we are facing to are global and complex as well: ecological issues, stability of markets, disarming, to name a few. However, our representative democracy is based on the bureaucratic processes implemented through rigid, slow and static forms of writing. [[Lévy, 1999](#), p. 60] Information technology is most often used only to support these bureaucratic processes — not to overcome them, but we should fully conceive the possibilities of new technologies in democracy as Thomas Jefferson in the quote suggests.

[Rodriguez et al. \[2007\]](#) describe a system called Smartocracy⁶, in which various collective decision-making algorithms are used by a testing community. Namely, *direct democracy* embodies simply the idea ‘one person/one vote’, so that the trust based social network is *not* used to calculate a collective decision and if a person does not vote, his/her vote is not taken into account. *Dynamically distributed democracy* (DDD) can handle the fact that the person sometimes does not vote — in such cases the vote is distributed proportionally according to network connections to other neighbours which the non-voting person trusts to. Finally, *proxy vote* is an extension of DDD such that the electorate is not generally equal in voting and a power of each person is directly proportional to the amount of trust s/he has. Each algorithm can be used for different kind of problems. For example, in case of an important issue that is understandable for all people (e.g. declaration of war), direct democracy can be used, whereas proxy vote can be used in a case of highly specific issue (e.g. nuclear waste disposal) that only

⁵ Depending on the conditions, it can be a *collective stupidity* instead.

⁶ See <http://www.smartocracy.net/>.

an expert subset of all people understand. Trust can be domain specific, so the person can leave her vote to neighbours only in some specific topics like health care or army. Domain specific trust relationships in Smartocracy have not been developed yet, though.

According to ideals of the Enlightenment, direct democracy formed by all enlightened citizens is the best way how to make decisions. *“The democratic ideal is not to the election of representatives but the greatest participation of the people in public life.”* [Lévy, 1999, p. 64] Rodriguez and Watkins argue for use of dynamically distributed democracy instead of direct democracy, because the latter is more error-prone with diminishing participation of citizens. Efficient utilization of knowledge dispersed throughout the electorate may help solve many of present complex problems. Moreover, such utilization may also improve a general feeling of each citizen, because s/he would not be treated as a *mass*. However, Smartocracy or systems mentioned in the section 1.3 on page 4 are either not intended for direct political action in the sense that they are merely a channel to express public opinion, or they are outside of the legal framework of our constitution that supposes representative democracy. Therefore, there is a gap between what is applicable and what is possible. We believe that system allowing self-organization of people into politically engaged groups overcomes this gap, because even if it is inside of our legal system, it is intrinsically based on wisdom of the self-organizing crowds and hence democratic in nature.

2.5 IMPLICATIONS ON THE SYSTEM DESIGN

If the self-organization is a way of modelling systems, it is possible also to design an artificial one. *“A key characteristic of an artificial self-organizing system is that structure and function of the system ‘emerge’ from interactions between the elements. The purpose should not be explicitly designed, programmed, or controlled.”* [Gershenson and Heylighen, 2003] In context of our objectives, this means that:

1. Our ontology and implemented system should not *a priori* conceive any particular political issue or organization. It should be as flexible as possible.
2. The whole system is the ontology, the web application and its users, so each of them has to have possibility to interact freely. Namely these kind of interactions should be possible: users with one another; users with the web application and users with the ontology.

First, because the ontology presents a conceptual space in which agents self-organize, this environment has to be as universal as possible. Namely, it must not to prescribe the content nor the topics of political life. Second, because needs and preferences of people can change in time, it is also crucial to allow them to modify the system so as to the whole community are in control and not a particular person, i.e. the author.

Knowing is not enough; we must apply!

— Goethe

This chapter is a general description of various knowledge technologies used in the system described in the second part of the thesis. Firstly, the semantic web vision is quickly introduced and after that its core technologies and standards are described. Secondly, the WordNet thesaurus for English is presented. Thirdly, the way of computing a similarity between concepts in WordNet is described.

3.1 THE SEMANTIC WEB

Nowadays web is simply a collection of interlinked hypertext documents, usually with multimedia content like images or embedded videos. Actually it is a global file system accessible via the HyperText Transfer Protocol ([HTTP](#)). One of the first emerged problem of this architecture was how to find some information. It resembled and to the much extent still resembles finding a needle in a haystack — just like sometimes it is hard to find a document in a large local file-system. This problem was partially solved by advanced web mining methods like PageRank or HITS, which use the structure of the web, i.e. the links between hypertext documents, to find out relevant documents in the sense of a set of keywords the user puts to the search engine. This way it is usually possible to find out relevant documents, although it is sometimes necessary to iteratively re-formulate a search query to broaden or to narrow the returned result set. The searching issue can be perceived as a sub-issue of the more general information re-use issue. In context of re-use, another problem occurs, because sometimes one is not searching a specific document, but rather some information that is not contained in any document, but it is scattered throughout the web. Finding the best price for a product is an example. It gets even more complicated if this information is not contained explicitly in any document on the web, but can be inferred from integrated view of several pieces of knowledge dispersed throughout the web.

These issue is highly unlikely solvable with today's architecture of the World Wide Web ([WWW](#)), because its foundational standards like HyperText Markup Language ([HTML](#)) or Cascading Style Sheets ([CSS](#)) were developed to deal only with an appearance and a structure of document and not a meaning of information contained in it. Therefore, The Semantic Web initiative¹ is continuously developing standards and technologies to represent, store, query and reason with knowledge in a largely distributed manner of the WWW. In the context of collaboration of people in the political domain, these possibilities are very important, because as should be apparent from the section [1.3](#), the number of

¹ See <http://www.w3.org/2001/sw/>.

```

1 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3
4 <http://www.ontopolis.net/resource/person#1> foaf:firstName "
   Arthur"^^xsd:string .
5 <http://www.ontopolis.net/resource/person#1> foaf:surname "Dent
   "^^xsd:string .
6 <http://www.ontopolis.net/resource/person#1> foaf:mbox <mailto:
   arthur.dent@ontopolis.net> .

```

Figure 2: Example of RDF in Turtle

collaboration and participation tools in political world is growing and this growth brings the danger of lack of interoperability of these system and thus it is possible that huge potential of social web applications for participation and collaboration of citizens in politics will be diminished by mutual incompatibility of these systems. For example, imagine the ordinary controversial issue before elections like the use of nuclear energy. Many people will manifest their opinions in social web applications, whether it is Facebook, Localocracy or anything else. How one could get a general overview of the opinion of the electorate?² How could be clear that people do or do not want to use nuclear energy? And even further, when people also propose solutions and their pros and cons, how could be these proposals integrated? We believe that without the use of the semantic web's technologies it is hardly possible. The next sections describe these standards and technologies.³

3.1.1 RDF

Resource Description Framework (RDF)⁴ is a standard for representing data on the semantic web in a machine-processable way. Every piece of data is represented in the form of *triple* statement, which consists of a *subject*, a *predicate* and an *object*. This formalism represents data in form of a graph, where each node is either an *URI*, a *blank-node* or a *literal*. Subject is represented by an Uniform Resource Identifier (*URI*) or a blank node. Predicate is represented by an URI and object can be represented by an URI, a blank-node or it can be a simple literal like string or date. Note that literal cannot be a subject. Nodes represented by an URI can also be called a *resource*. Blank node is a special type of anonymous resource that is not identified by an URI. RDF can be serialized into various format like eXtensible Markup Language (*XML*), Notation 3 (*N₃*) or Terse RDF Triple Language (*Turtle*)⁵. The figure 2 contains a set of three triple statements formalizing the knowledge that person with first name Arthur (line 4) and surname Dent (line 5) has e-mail address arthur.dent@ontopolis.net (line 6).

Used predicates are defined in ontologies that are more discussed in chapter 4 on page 25.

² Note that only the electorate active on the web is considered. Implications of the so-called "digital-divide" is not discussed in this thesis.

³ Note that for the sake of brevity, we focus only on aspects needed to understand further topics of the thesis. For complete descriptions, please see cited sources.

⁴ See <http://www.w3.org/RDF/>.

⁵ Both RDF/XML (<http://www.w3.org/TR/rdf-syntax-grammar/>) and N₃ (<http://www.w3.org/DesignIssues/Notation3.html>) are W₃C specifications. Turtle is a non-standard, but widely adopted, human-readable serialization of RDF (<http://www.w3.org/TeamSubmission/turtle/>).

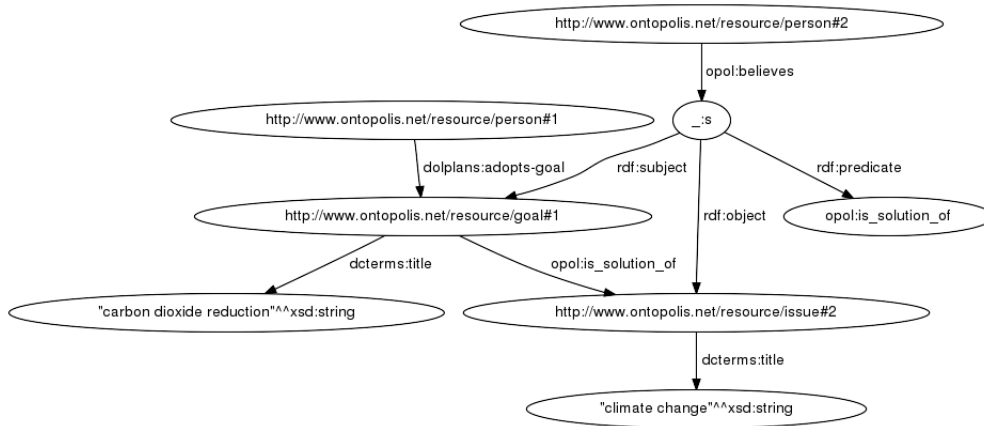


Figure 3: Diagram of RDF reification

```

1 @prefix dolplans: <http://www.loa-cnr.it/ontologies/Plans.owl#>.
2 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
3 @prefix opol: <http://www.ontopolis.net/ontopolis/o.1/> .
4 @prefix dcterms: <http://purl.org/dc/terms/> .
5 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
6
7 <http://www.ontopolis.net/resource/person#1> dolplans:adopts-
   goal <http://www.ontopolis.net/resource/goal#1> .
8 <http://www.ontopolis.net/resource/goal#1> dcterms:title "Carbon
   Dioxide Tax"^^xsd:string .
9 <http://www.ontopolis.net/resource/goal#1> opol:is_solution_of <
   http://www.ontopolis.net/resource/issue#2> .
10 <http://www.ontopolis.net/resource/issue#2> dcterms:title "
   Climate Change"^^xsd:string .
11 _:s rdf:subject <http://www.ontopolis.net/resource/goal#1> .
12 _:s rdf:predicate <http://www.ontopolis.net/ontopolis/o.1/is_
   solution_of> .
13 _:s rdf:object <http://www.ontopolis.net/resource/issue#2> .
14 <http://www.ontopolis.net/resource/person#2> opol:believes _:s .

```

Figure 4: RDF reification in Turtle

The first two lines have a literal as an object. Literal can be untyped or typed. Standard XML Schema Definition (XSD) build-in datatypes⁶ are used. In this example, both literals are typed as a string.

Besides these triple statements which simply assign some predicate to a subject, RDF also provides a way to make a statement about the statement which is called *reification*. For instance, if Arthur Dent said that a goal is a solution of some issue, someone else could want to declare that s/he believes to Arthur's statement, i.e. that accomplishing the goal the issue will be solved. The figure 4 illustrates the situation, where the person#1 states that the imposition of a special tax to carbon dioxide producers will solve the problem of climate change (line 9). In this example, another person#2 declares the belief in this statement (line 14). Note that `_:s` (line 11-14) is a blank node, which is a reification (line 11-13) of the statement of person#1 about the solution of the issue. The figure 3 provides graphical and probably more understandable view of this set of RDF statements.

6 See <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#built-in-datatypes>.

Jena is more discussed in the section 5.2.2 on page 41.

In addition to serializing RDF to one of its formats, it is also possible to store an RDF graph to a persistent store. Sesame⁷, OWLIM⁸, Oracle database⁹ or Jena¹⁰ can be an example of such a store. It is either backed by store's native files (one of the options for Sesame), or there is a relational database on the backend (Jena's SDB storage). Both options has its benefits as well as its pitfalls and it is not a purpose of this thesis to discuss it. RDF stores usually provides an API for accessing and manipulating underlying data and some of them allows also access via SPARQL Protocol and RDF Query Language (SPARQL) that is described in the section 3.1.3.

RDF graph can also have a name defined by an URI and then it becomes a *named graph*, which is a set of *quads* instead of triples. The fourth element is the URI of the named graph. This is useful for example when it is necessary to differentiate the origin of the data, or to make a statement about a set of statements and not about only one as it is possible by reification.

RDFS

As RDF is a datamodel, another language is needed to express meaning of data on the semantic web. RDF Schema (RDFS)¹¹ is a simple language for definition of RDF schemas that allows one to define taxonomies and RDF vocabularies, and thus RDFS consists of concepts like `Class`, `Resource` or `Literal`. Because we have chosen Web Ontology Language (OWL) instead of RDFS, we are not describing here this language. Note that `rdfs:Literal` is a class of literal values, e.g. strings, dates, integers, etc.

3.1.2 OWL

OWL¹² is a standard language for representing knowledge on the semantic web. It allows to "*write explicit, formal conceptualizations of domain models*". [Antoniou and van Harmelen, 2008, p. 114] The OWL contains vocabulary to define classes and their properties. It is also possible in OWL to specify this knowledge more precisely and declare disjointness of classes (e.g. class of all men is *disjoint* with the class of all women); symmetricity of a property (relation) (e.g. if person *A* is a relative of person *B*, then person *B* is also a relative of person *A*); domain and range of a property (e.g. a property denoting relationships has to have persons in both its domain and its range); or cardinality restrictions (e.g. that a person can have at most 2 parents).¹³ There are three flavours of OWL (*ibid*, p. 117):

OWL Full Contains all languages' primitives and is fully upward-compatible with RDF and RDFS. Disadvantage of this powerfullness is undecidability.

⁷ See <http://www.openrdf.org/>.

⁸ See <http://www.ontotext.com/owlim/>.

⁹ See http://www.oracle.com/technology/tech/semantic_technologies/index.html.

¹⁰ See <http://jena.sourceforge.net/>.

¹¹ See <http://www.w3.org/TR/rdf-schema/>.

¹² We used OWL version 1 in our work, but the OWL 2 has become available recently.

¹³ Note that this list of OWL's capabilities is only illustrative. The complete language reference can be found at <http://www.w3.org/TR/owl-ref/>.

OWL DL Restricts how the language primitives of RDF and OWL-Full may be used in order to preserve decidability. *DL* in the name means *Description Logic*, and thus OWL DL corresponds with description logic $\mathcal{SHOIN}^{(D)}$.

OWL Lite Contains a subset of OWL DL, so it is easier to understand (for users) and to implement (for tool builders).

With ontology defined in one of these flavours, it is possible to reason over it. Namely, the following typical tasks can be done:

- “check for consistency of the ontology and the knowledge”
- determine intended and “check for unintended relationships between classes”
- “automatically classify instances in classes” (*ibid*, p. 115)

There are several reasoners available.¹⁴ Sometimes, they are integrated directly into an RDF store. Others can be used either stand-alone, or they can be plugged into the third-party application.¹⁵

To assign a class to a resource, RDF defines the `rdf:type` property. For example, to state that `person#1` is an instance of class `foaf:Person`, the `<http://www.ontopolis.net/resource/person#1> rdf:type foaf:Person` statement should be added into the knowledge base.¹⁶

In the world of relational databases, it is assumed that what is unknown, i.e. what is not contained in a database, is **false** — this is called a Closed World Assumption (*CWA*). On the contrary, OWL adopts an Open World Assumption (*OWA*): “a statement cannot be inferred to be false on the basis of a failure to prove it”. [Sirin and Tao, 2009]. For example, consider `foaf:knows` property that has a `foaf:Person` in both its range and domain. These three triples:

```
<http://www.ontopolis.net/resource/goal#1> rdf:type doledns:goal .
<http://www.ontopolis.net/resource/person#1> rdf:type foaf:Person .
<http://www.ontopolis.net/resource/person#1> foaf:knows <http://www.ontopolis.net/resource/goal#1> .
```

will not cause an error during validation. Because of *OWA*, it is inferred that `goal#1` is also an instance of `foaf:Person`, instead.

Moreover, OWL does not adopt the so-called *Unique Names Assumption (UNA)* which would cause OWL tools to treat two resources with different identifiers as distinct objects. (*ibid*) Because of *UNA*, if for instance an example:has_mother property has a cardinality restriction so that one person has to have exactly one mother, these two triples:

```
<http://www.ontopolis.net/resource/person#1> example:has_mother <
  http://www.ontopolis.net/resource/person#2> .
<http://www.ontopolis.net/resource/person#1> example:has_mother <
  http://www.ontopolis.net/resource/person#3> .
```

will not cause an error. It is inferred that `person#2` and `person#3` are equal, instead.

Difficulties caused by OWA and UNA and their solutions are discussed in the section 5.2.3 on page 45.

¹⁴ See http://en.wikipedia.org/wiki/Semantic_reasoner.

¹⁵ Besides various native APIs, there exists a standard interface for integration with reasoners through XML called DIG. For more information, see <http://dig.sourceforge.net/>.

¹⁶ Note that the `rdf:type` is in some notations replaced by keyword `a`.

```

PREFIX dolplans: <http://www.loa-cnr.it/ontologies/Plans.owl#> .
PREFIX opol: <http://www.ontopolis.net/ontopolis/0.1/> .
SELECT ?goal
WHERE {
  <http://www.ontopolis.net/resource/person#1> dolplans:adopts-
    goal ?goal .
}

```

Figure 5: Simple SPARQL statement

3.1.3 SPARQL

For accessing RDF data on the semantic web, the [SPARQL \[Prud'hommeaux and Seaborne, 2008\]](#) is used. It is a W3C recommendation and for the semantic web it is as important standard as Structured Query Language (SQL) is for relational databases. In fact, syntax of SPARQL is very similar to SQL. As it is apparent from its name, SPARQL specifies not only a query language, but also specifies the protocol for querying remote RDF stores via HTTP.

Simple SPARQL query to an RDF graph described in figures 3 and 4 returning all goals that person#1 adopts is listed in figure 5.

The WHERE clause consists of a set of *triple patterns* that have the ordinary subject-predicate-object form. The pattern can contain a variable that will be bound to a value according to queried RDF graph and according to other triple patterns (if any). A set of triple patterns is called a *graph pattern* and SPARQL is based on matching these graph patterns. (*ibid*)

SPARQL defines several *query forms*. Except **SELECT**, following forms are available (*ibid*):

- ASK** Returns whether a graph pattern has a solution (true) or not (false).
- CONSTRUCT** Returns an RDF graph described by a template and a graph pattern.
- DESCRIBE** Returns an RDF graph described by a graph pattern and potentially additional information about matched resources. Exact form of output depends on used SPARQL query processor.

SPARQL also provides additional language elements for restricting number of solutions returned (LIMIT modifier), start of the solution (OFFSET), guaranteeing the uniqueness of some variable (DISTINCT) or ordering a result set according to some variable (ORDER BY). Use of all of these elements is similar to their counterparts in SQL. It is also possible to select some subset of a graph pattern as an optional (OPTIONAL) or to select a union of results of two distinct graph patterns (UNION). As for named graphs, the GRAPH keyword is available to specify which graph should be queried.

```

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

INSERT {
  <http://www.ontopolis.net/resource/person#1> foaf:firstName "
    Arthur"^^xsd:string .
  <http://www.ontopolis.net/resource/person#1> foaf:surname "
    Dent"^^xsd:string .
  <http://www.ontopolis.net/resource/person#1> foaf:mbox <mailto:
    arthur.dent@ontopolis.net> .
}

```

Figure 6: Simple SPARUL statement

3.1.4 SPARUL

SPARQL/Update Language ([SPARUL](#)) [[Seaborne and Manjunath, 2008](#)] is a language for updating RDF graphs. It was developed by HP¹⁷ and it is not a W₃C standard currently. According to the specification, it allows to:

- **insert new triples** to an RDF graph
- **delete triples** from an RDF graph
- **modify triple** in an RDF graph
- perform a **bulk update** of an RDF graph
- **create a named graph** in a store
- **delete a named graph** from a store

Roughly speaking, SPARUL is for SPARQL what Data Manipulation Language ([DML](#)) is for SQL. Simple SPARUL example that adds the set of triples shown in figure 2 on page 14 is depicted in the figure 6.

3.2 WORDNET THESAURUS

WordNet¹⁸ is a lexical database for English.¹⁹ The vocabulary of a language is defined as “a set W of pairs (f, s) , where a form f is a string over finite alphabet, and a sense s is an element from a given set of meanings”. [[Miller, 1995](#)] An element of W is a word of the language. In WordNet, many semantic relations are used between words, like synonymy, antonymy or hyponymy, just to name a few. However, “*synonymy is WordNet’s basic relation, because WordNet uses sets of synonyms (synsets) to represent word senses*”. (*ibid*)

The synonymy is a relation between two different words that share at least one sense in common. The hyponymy is a transitive relation between synsets, where one synset (a hyponym) has its semantic range fully covered by another synset (a hypernym). The word “oak tree” is

¹⁷ See <http://www.hpl.hp.com/semweb/>.

¹⁸ See <http://wordnet.princeton.edu/>.

¹⁹ Project EuroWordNet develops similar databases for several European languages including Czech. See <http://www.i11c.uva.nl/EuroWordNet/>.

a hyponym of “plant”, which is its hypernym. The inverse relation of hyponymy is hypernymy. This relation organizes nouns into a hierarchy, therefore it is possible to compute a semantic similarity based on a distance between two words. Section 3.3 covers this topic in more detail.

WordNet is distributed usually in the form of specially formatted files and there exist several libraries²⁰ or client applications to work with these files. There also exist a conversion into RDF/XML format.²¹ This conversion is available in two forms: *basic* and *full*. Both contains a set of synsets, but whereas the *full* version provides additional information about words and word senses (like antonymy relation or derivational relatedness), the *basic* provides good support for disambiguation process with less memory footprint. In addition to the file containing the set of synsets, each version has its own OWL schema and another file containing either a set of word senses and words (the *full* version) or a set of sense labels (the *basic* version). The file with sense labels contains a set of strings for each synset, e.g. for the synset `synset-living-thing-noun-1` it defines these two sense labels: `animate thing` and `living thing`. Except these three files, additional files can be used to add relations like hyponymy. Some of these relations are defined between word senses (antonymy, for instance) and cannot be used in the *basic* version, however.

3.3 SEMANTIC SIMILARITY

To support matching of similar-minded people in politics, it is necessary to use a similarity measure between descriptions of their proposed solutions and goals. There exists many similarity measures such as mutual information[Rijsbergen, 1979, p. 27], Dice coefficient, cosine coefficient or Jaccard index (*ibid*, p. 25), but Lin [1998] claims they are either tied to a particular application or assumes a particular domain model. For example, Dice or Jaccard coefficient are applicable only when objects are represented as numerical feature vectors. Another claim is that underlying assumptions of these similarity measures “are often not explicitly stated”. Therefore, author comes with an information-theoretic definition of similarity that is applicable everywhere, where the domain has a probabilistic model. This similarity measure, let us called it sim_{lin} hereinafter, is based on three basic intuitions (*ibid*):

1. *The similarity between A and B is related to their commonality. The more commonality they share, the more similar they are.*
2. *The similarity between A and B is related to the difference between them. The more differences they have, the less similar they are.*
3. *The maximum similarity between A and B is reached when A and B are identical, no matter how much commonality they share.*

In addition to these intuitions, author makes also six additional assumptions based on these intuitions. We point out here only the first and the second assumption (*ibid*):

1. *The commonality between A and B is measured by $I(\text{common}(A, B))$, where $\text{common}(A, B)$ states the commonalities between A and B; $I(s)$ is the amount of information contained in a proposition s.*

²⁰ As for Java, JWNL is an example. See <http://jwordnet.sourceforge.net/>.

²¹ See <http://www.w3.org/2006/03/wn/wn20/>.

2. The difference between A and B is measured by $I(\text{description}(A, B)) - I(\text{common}(A, B))$, where $\text{description}(A, B)$ is a proposition that describes what A and B are.

Finally, the similarity theorem is derived:

$$\text{sim}_{\text{lin}}(A, B) = \frac{\log P(\text{common}(A, B))}{\log P(\text{description}(A, B))}$$

This formula is applicable to similarity between objects represented by ordinal values, feature vectors, strings, words and even concepts in a taxonomy, which is of our particular interest. If treating WordNet synsets with hyponymy/hypernymy relations as a taxonomy, the similarity between two synsets is

$$\text{sim}_{\text{lin}} = \frac{2 \times \log P(C_0)}{\log P(C_1) + \log P(C_2)},$$

where $P(C_i)$ is a probability, that a randomly selected object belongs to C_i and C_0 is the most specific class that subsumes both C_1 and C_2 . (*ibid*)

For example²², the figure 7 is a fragment of WordNet, where each number under the concept C is $P(C)$ and edges denote hyponymy/hypernymy relation. If $C_1 = \text{"hill"}$, $C_2 = \text{"coast"}$ and $C_0 = \text{"geological - formation"}$, the similarity between "hill" and "coast" is

$$\text{sim}_{\text{lin}}(\text{hill}, \text{coast}) = \frac{2 \times \log(0.00176)}{\log(0.0000189) + \log(0.0000216)} \doteq 0.59.$$

Author compares sim_{lin} with other commonly used similarity measures for WordNet and concludes that sim_{lin} performs slightly better than the others (*ibid*).²³

²² This example is based on the original example of WordNet similarity measure in the cited article.

²³ The correlation of sim_{lin} with assessments made by human subjects was 0.834, whereas the second best was 0.803.

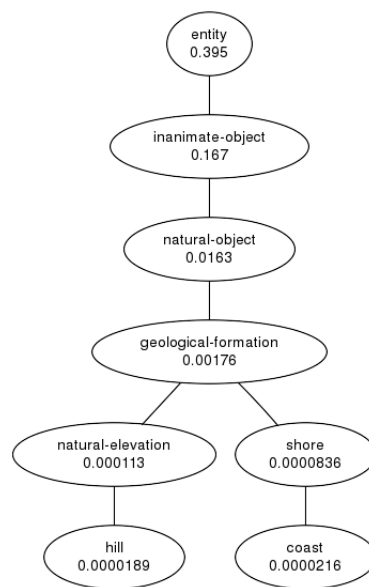


Figure 7: Fragment of WordNet

Part II

ONTOPOLIS PROJECT ACHIEVEMENTS

In framing an ideal we may assume what we wish, but should avoid impossibilities.

— Aristotle

This chapter describes the Ontopolis schema or simply *OPOL*, which has been developed as a part of this thesis. It is used as a schema for *all* data in the system, whose architecture is presented in the next chapter. The main purpose of this ontology can be characterized by the following competency questions [Gruninger and Fox, 1994]:

- What are actual political issues that people are interested in?
- How are these issues interrelated?
- What are possible solutions of these issues?
- Which of these solutions are more worthy of attention?
- Given one particular proposed solution of some issue, what are similar solutions?
- Who is interested in similar political topics as a given user?

Another motivation for creation of this ontology is to provide a vocabulary to be shared between e-democracy and/or e-participation sites. Various popular ontologies are re-used (i.e. imported) in OPOL in order to be as compatible with other systems as possible and each of them is described in the following text. Note that we are particularly focused on parts of these ontologies that are directly related to the topic of this thesis and for this reason large portion of each ontology is omitted. In context of each imported ontology we also provide a description of our extensions (if any) and its use in the system.

The whole schema can be found in the appendix A on page 63.

4.1 INCORPORATING DCTERMS

Dublin Core Metadata Initiative (DCMI) is an organization developing metadata standards for description of information resources across domains. This organization developed the set of fifteen terms called DCMI Element Set¹, or simply *DCMES*, which is also an ISO standard. DCMI Terms², or simply *DCTerms*, is another schema by DCMI containing all metadata terms maintained by this organization. Properties in DCMES have not their domains and ranges defined. DCTerms contains fifteen new properties with the same name, but with ranges and properties properly specified where possible. These new properties are subproperties of corresponding ones from DCMES. In spite of the

¹ See <http://dublincore.org/documents/dces/>.

² See <http://dublincore.org/documents/dcmi-terms/>.

| NAME | DOMAIN | RANGE |
|-------------|--------|--------------|
| identifier | - | rdfs:Literal |
| description | - | - |
| date | - | rdfs:Literal |
| title | - | - |

Table 1: Dublin Core metadata terms used in OPOL

fact it is possible to use both of these standards, “implementers are encouraged to use the semantically more precise *dcterms*: properties, as they more fully follow emerging notions of best practice for machine-processable metadata”.^[DCTerms] In OPOL, we are using DCTerms properties listed in the table 1.

4.2 INCORPORATING FOAF

Friend Of A Friend (FOAF)³ [FOAF] schema is used for description of persons, users, their groups and mutual relationships. Some elements of the FOAF vocabulary were already presented as a part of the figure 2 on page 14. In OPOL, we use classes and properties depicted in the figure 8.⁴ The edge with empty arrow at its head signs the inheritance, whereas the one with full arrow at its head connects the domain of the property, whose name is a label of a given edge, with its range (a head of a given edge). One of the differences between FOAF and ordinary approach to storing information about people, e.g. in relational database, is the split of `Person` and `OnlineAccount` concepts. The latter is a class describing some kind of *service* provided by a system in the Internet, e.g. a web application. The former is a subclass of `Agent` that describes all entities that are able to *perform an action*. This split is both more correct and useful, because a person is only one, but s/he can use several online services — s/he can be a *user* of various systems. Property `holdsAccount` connects the agent to its online account. In OPOL, two such an *agentive* classes are particularly used: `Person` and a politically engaged group — `opol:PEGroup`, which is a group of persons who have a joint set of goals in politics. Because a politically engaged group, e.g. a political party, can be considered as an agentive object, it is a subclass of `Agent`. As well as `Person` has its counterpart in `User`, `opol:PEGroup` has its counterpart in `opol:PEUsergroup` that is a set of users (i.e. online accounts), which are used by the persons that are members of corresponding politically engaged group. Further, `opol:MailBox` class is introduced to represent an e-mail address of person.⁵ Social relationships between persons can be represented by `knows` property. The semantics of this relation is intentionally broad

Representation of goals is a topic of the section 4.5 on page 32.

³ See <http://www.foaf-project.org/>.

⁴ Note that for the sake of clarity we omitted the `foaf` prefix in those classes and properties, which are a part of FOAF.

⁵ Property `mbox` has the `owl:Thing` as its range.

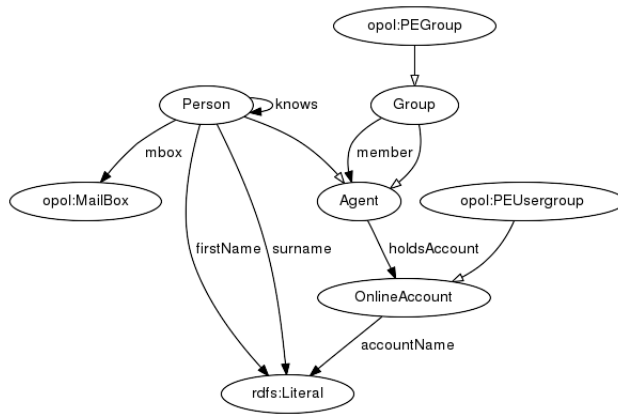


Figure 8: FOAF vocabulary used in OPOL

and covers all relations from “person X know that person Y exists” to “person X is a father of person Y”.⁶

4.3 INCORPORATING SIOC

Content generated by users on the web is constantly growing and it plays a significant role in community-based supporting forums and wikis in the open-source community, for instance. However, some information is useless quite often, because other information necessary for full comprehension is missing at a given forum or wiki, but it is available somewhere else on the web. Therefore, it is needed to interlink these online communities based around various forums, blogs, wikis, etc. This is the main purpose of Semantically-Interlinked Online Communities (SIOC)⁷ ontology.[SIOCC]

SIOC is to a great extent complementary to FOAF, because whereas FOAF allows to represent relationships between people and their profiles in various systems on the web, SIOC allows to represent relations between people and *content* they have created. Moreover, SIOC covers also vocabulary to represent a topic of a content, online place, where it was created (concrete blog, wiki, etc.) and relations to another content. Basic vocabulary is defined in a *core* SIOC ontology. The general overview of the core ontology is provided in the figure 9. It represents a content in a way where a content item (e.g. a Post) is a part of a Container (e.g. a discussion Forum), which belongs to a Space (e.g. a web Site). This formalism can be easily extended and in fact one of three currently existing extension modules provides additional types of these general concepts. These three modules are:

- access** Contains a vocabulary for definition of users rights and permissions in online services.
- types** Contains various extensions of basic terms from the core ontology.

⁶ For this reason, an extension of FOAF for description of relationships has been created. For more information, please see <http://vocab.org/relationship/.html>.

⁷ For general overview, see <http://sioc-project.org/>. The specification of the ontology can be found at <http://rdfs.org/sioc/spec/>.

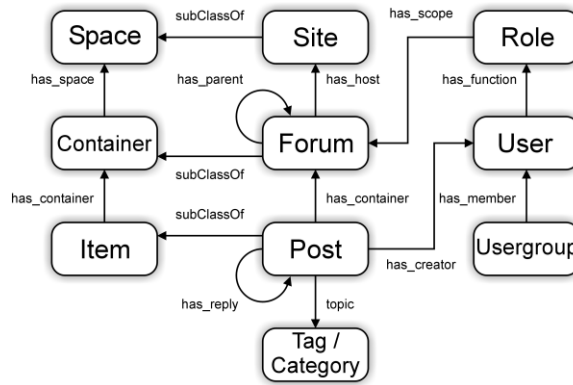


Figure 9: Overview of SIOC core ontology (source: [SIOCC])

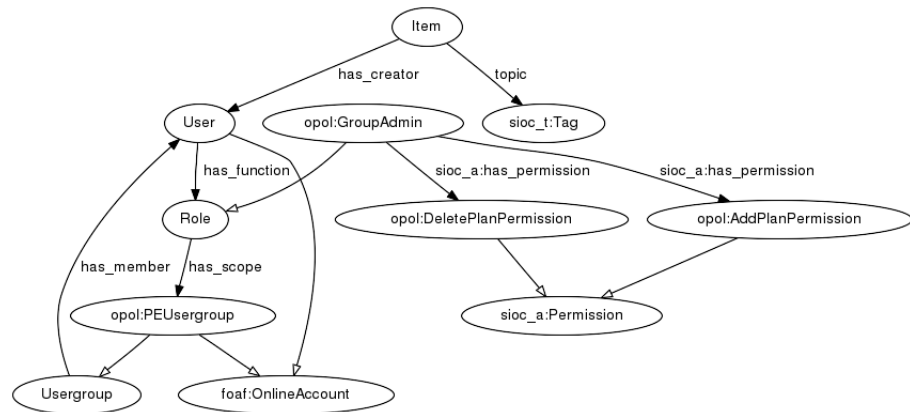


Figure 10: SIOC vocabulary used in OPOL

services Contains extensions of the core ontology for description of web services available on a site.

First two of these additional modules together with the core ontology are used in OPOL.

The figure 10 illustrates SIOC vocabulary used in it. Prefix `sioc_c` is omitted in those classes and properties, that are defined in the *core* ontology. Prefixes `sioc_t` and `sioc_a` represent modules *types* and *access*, respectively. Every content item can has its creator, which is an instance of class `User`. It can also has a topic, which is in our case described by tags, hence the `Tag` class is used. We have already encountered with `opol:PEUsergroup` class in the previous section. In context of SIOC, it is necessary to point out this class is both a subclass of `foaf:OnlineAccount` for reasons discussed in the section 4.2 and `Usergroup`, because it is defined as “a set of *User* accounts whose owners have a common purpose or interest”. [SIOCC] Users can have various functions in scope of `opol:PEUsergroup`. Function of a user is determined by his role and to a role, particular permissions are assigned to. Presently, only one role is defined in OPOL — `opol:GroupAdmin` that is typically assigned to a user who is a creator of the group. Note that this is a *user* role and not a role of an *agent*.

Usage of tags is a subject of sections 4.5 and 5.2.5 on page 47.

4.4 INCORPORATING DOLCE FOR REPRESENTATION OF POLITICAL PROGRAMS

In ontological engineering, the special role is played by so-called *foundational ontologies*⁸, which specify general (foundational) concepts shared across all domains. Use of foundational ontology helps an ontology engineer to keep focus on a given domain while allows one to align created ontology with another one, if both use general concepts from a foundational ontology. One of the foundational ontologies⁹ for the semantic web is Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [DOLH]. It is intended to be a starting point for various extension modules. DOLCE, as it is apparent from its name, “has a clear cognitive bias, in the sense that it aims at capturing the ontological categories underlying natural language and human commonsense”. [Gangemi et al., 2002] This is very important to our work, because it allows us to represent various concepts and relations in political reality like goals of a political candidate “in a post-hoc way, reflecting more or less the surface structure of language and cognition”. (ibid)

We have chosen the simplified version of DOLCE, the *DOLCE-Lite*, with the following modules:

Plans Module defining concepts for representation of plans, goals, tasks and appropriate relations. It depends on DnS (see below).

ModalDescriptions Plug-in to DnS providing modal relations and concepts for description of commitments, promises, etc.

In fact, these two modules import others. Complete overview of dependencies is provided in the figure 11. For the sake of brevity, we are not describing here all these modules, except the *ExtendedDnS*, which is noticeably a key component of them. Descriptions and Situations (DnS) is “an extension of DOLCE whose main intent is enabling the ontological talk about non-physical, social and especially knowledge objects. The rationale is that the properties that we attribute to entities are entities as well, and we can treat them as ‘knowledge’ or ‘information’ objects. A description is a non-physical object (in particular, it is a non-agentive social object), which represents a conceptualization, hence it is generically dependent (GD) on some agent, and which is also social, i.e. communicable”. [Gangemi et al., 2004] For example, theory, plan and goal are all descriptions. “A situation has to satisfy a description . . . , and it has to be the setting for at least one entity from the ground ontology” (ibid), i.e. DOLCE. A model of a theory, a plan execution or a desired state can all be examples of situations. *ExtendedDnS* is the DnS ontology with additional vocabulary for social reification.

We are not using situations in OPOL currently, although it is an interesting topic, because it points out to a concrete real-world problems and happenings.¹⁰ We do not need this kind of objects to answer competency questions mentioned at the beginning of this chapter, however.

⁸ Also called *upper* or *top-level* ontologies.

⁹ A list of foundational ontologies available can be found at [http://en.wikipedia.org/wiki/Upper_ontology_\(information_science\)](http://en.wikipedia.org/wiki/Upper_ontology_(information_science)).

¹⁰ For instance, the goal-situation is a fulfilling of some goal (and corresponding promise) of a politician.

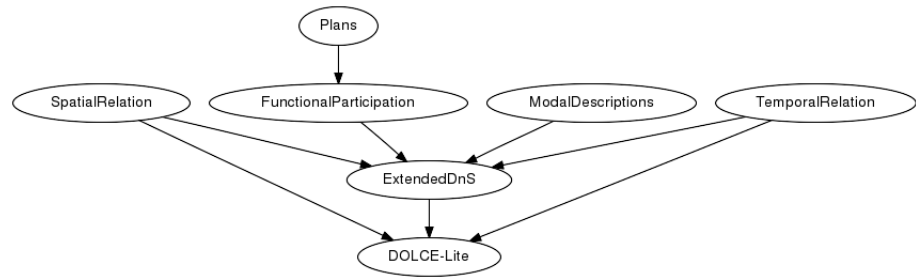


Figure 11: Dependencies between used DOLCE modules

4.4.1 Representing political programs

Our use of DOLCE and related concepts of OPOL is illustrated in the figure 12.¹¹ Van Attevelde [2008, p. 153] discuss various ways to represent dynamic (i.e. changing in time) political roles and argue for creation of an *adjunct instance* for each role played by a person, because it allows easier reasoning and querying. Similar way is chosen in DOLCE, therefore we use it in OPOL too. An agent, i.e. a person or a group, can play one of two political roles: it is either a `opol:Supporter` or a `opol:PoliticalCandidate`. New instance is created for each role played by an agent in a plan (see below). Political candidate is a role of the politically engaged agent, who wants to be elected in a poll. Supporter is also a role of the politically engaged agent, but who has generally not so strong political ambitions. Supporter only declared his/her support to the candidate by the `opol:Following` commitment that means the supporter is a follower of the candidate and it is expected the follower will not behave counter to this commitment.

Political issues are naturally hierarchical. There are several ways how to represent this hierarchy. Various possibilities are discussed in [van Attevelde, 2008, p. 155] and finally it is argued there for creation of hierarchy of instances. Similar to van Attevelde's approach, we defined the `opol:subissue_of` property, which is a subissue of `skos:broader`.¹² (*ibid*, p. 156) Each problem to solve in politics is described by an instance of `opol:PoliticalIssue`. Issue can be a sub-issue of another issue. The political candidate defines a plan around an instance of `opol:SolutionPlan` that is a subclass of `doledns:plan`. "A plan is a description that defines or uses at least one task ... and one agentive role or figure, and that has at least one goal as a part". (*ibid*, p. 32) In OPOL, the solution plan defines at least one instance of `opol:measure`, which is a sub-class of `doledns:task`, and that is assigned to the political candidate. Solution plan can also define an instance of `opol:Support` that is a task assigned to a supporter. Solution plan has to have also one goal that is declared to be a solution of the political issue. The agent who plays the political candidate role defined by the plan adopts the plan and its goal as well. When agent adopts the plan, it makes a promise (i.e. instance of `dolmd:promise`), which can be perceived as a commitment to realize the plan.

¹¹ Note that `opol`, `dolmd`, `doledns` and `dolplans` are prefixes for Ontopolis, ModalDescriptions, ExtendedDnS and Plans ontologies, respectively.

¹² Simple Knowledge Organization System (SKOS) ontology is intended to help to create classification schemes, thesaurus, etc. Property `skos:broader` should be read as "has broader". See <http://www.w3.org/2004/02/skos/>.

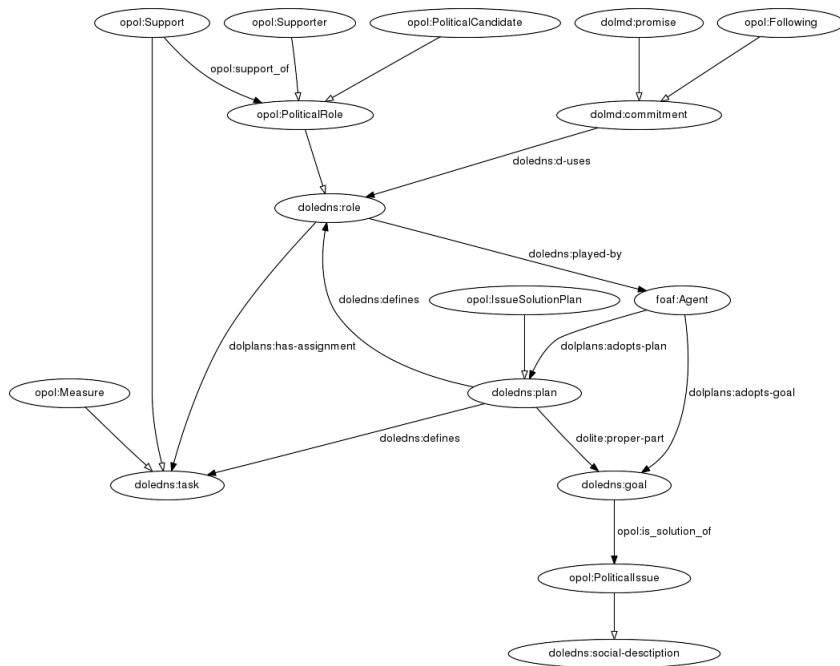


Figure 12: Elements of DOLCE used in OPOL

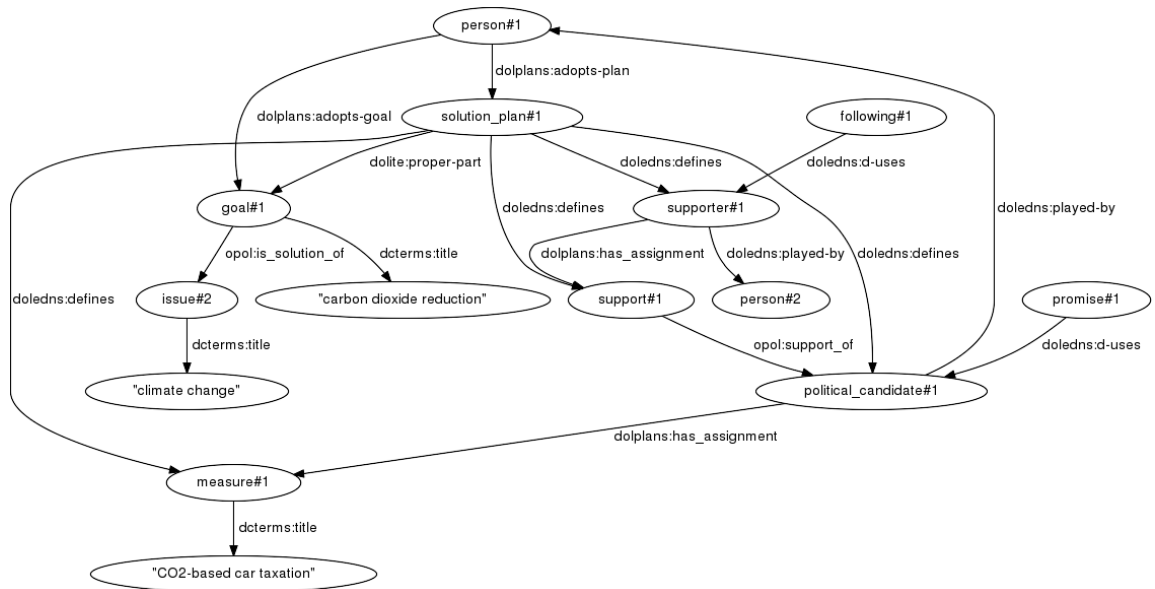


Figure 13: Example of solution plan

The figure 13 is a modified version of the figure 3. Vocabulary described in the previous text is used to fully describe the political program of person#1.¹³ The program consists of one plan, which is the solution of “climate change” issue. To solve this issue, person#1 suggests to impose a special tax on cars based on their production of CO₂. Because s/he have created the plan, s/he plays the role of a political candidate. By becoming a candidate, s/he states the promise to achieve the goal, which is a part of the solution plan. To declare affinity to this plan and/or candidate, another person#2 became a follower of person#1 and thus s/he stated a corresponding commitment, i.e. following#1.

4.5 REPRESENTATION OF WORD SENSE DISAMBIGUATION

To express meaning of resources on the WWW, *tagging* has been widely adopted among social web applications. One of the advantages of tagging is its simplicity. On the other hand, the ubiquitous polysemy of natural language is its pitfall. Mika [2005] argues for a unified view of social networks and semantics. He shows how it is possible to extract taxonomies of concepts, their clusters or how to extract a network of people with similar interests. These structures can be used for example to recommend to a user another one with similar interests. This approach needs certain critical amount of data. However, when a social application is rather empty, we are facing the chicken-egg problem, because users are not motivated to create tags, because it does not provide them any advantage yet, but to provide it (i.e. an intelligent recommendation) the critical mass of tags is needed. To overcome this problem, one of the possible solutions is to disambiguate tags in the time they are assigned to a resource in order to determine tags’ meanings and thus enable the intelligent recommendation.

In OPOL, we have chosen a WordNet thesaurus for word-sense disambiguation. As it is described in section 3.2 on page 19, every word sense is represented as a synset (a set of synonyms). Let the {“environment”, “protection”, “greenery”} to be a set of tags describing a solution plan. The following WordNet’s definitions illustrates difficulties of determining a sense of these words:

ENVIRONMENT 1. the totality of surrounding conditions; 2. the area in which something exists or lives

PROTECTION 1. the activity of protecting someone or something; 2. a covering that is intend to protect from damage or injury; 3. defense against financial failure; financial independence; 4. the condition of being protected; 5. kindly endorsement and guidance; 6. the imposition of duties or quotas on imports in order to protect domestic industry against foreign competition; 7. payment extorted by gangsters on threat of violence

GREENERY 1. green foliage

A synset is assigned to each *usage* of tag during the process of disambiguation. There are several ways how to represent this in the context of

¹³ Note that for the sake of brevity we omitted resource prefixes (i.e. <http://www.ontopolis.net/resource/>) and suffixes of string literals (i.e. `^^xsd:string`) in further text.

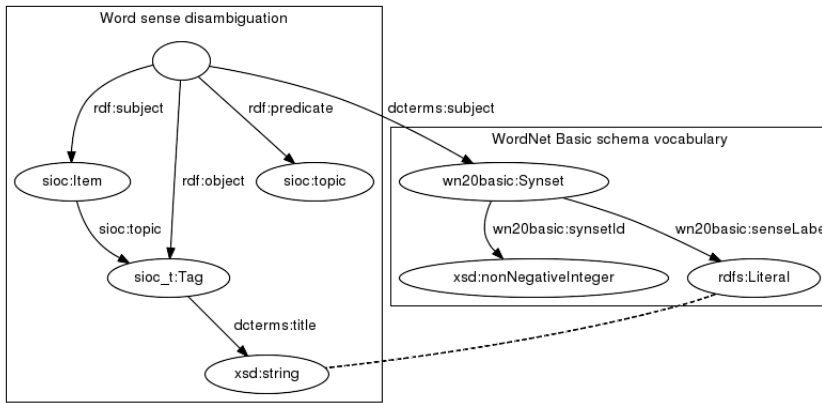


Figure 14: WordNet Basic schema and its use in disambiguation

previously discussed ontologies. First, it is possible to treat each occurrence of tag as an instance of a special class (*TagOccurrence*, for example) and to relate it to an instance of `sioc_t:Tag` (i.e. the “environment”, for example) and to a corresponding synset determining an actual meaning of this tag occurrence. The second way is to relate a corresponding synset to a reified statement that relates an instance of `sioc:Item` to an instance of `sioc_t:Tag`. Both options has their pros and cons: the former is more practical during querying, but it brings the need of new class and property, whereas the latter is less intrusive, but also less practical during querying. Taken these two options into account, we have finally chosen representing a disambiguation by reification as it is depicted in the figure 14. On the right part of the picture there is a fragment of WordNet Basic schema. Each synset has unique identifier and it is related to one or more strings by `wn20basic:senseLabel` property. These strings represent a lexical form of words that belong to the synset. On the left part of the picture a representation of tags related to the synset is depicted. Each tag has exactly one title. Note that `dcterms:title` property has undefined range but in context of OPOL it is used always with a string literal in its range. During disambiguation this title is matched with one or more words (depicted as a dashed association) and the most suitable synset is determined. Then it is related by `dcterms:subject` property to a reification of the statement (depicted as a blank node) relating the item to the tag.

Disambiguation and similarity algorithms are described in the section 5.2.5 on page 47.

Similarity relations

Disambiguated tags are used to determine overall similarity between two tagged items. Because this operation is resource-consuming, its result is saved into the store. The left part of the figure 15 presents a part of the OPOL that is used to represent the similarity relations. An instance of `opol:SimilarityInfo` is always related to exactly two tagged items and, of course, the value of their overall similarity. The right part of the figure then presents an example of usage of the vocabulary.

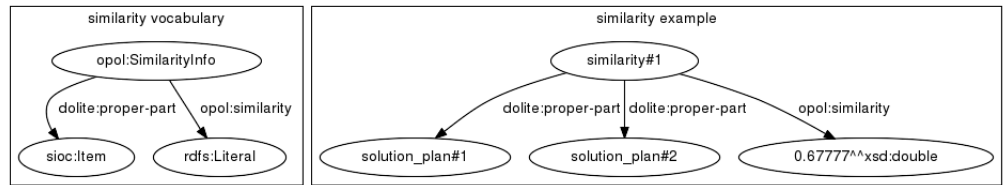


Figure 15: Similarity relations representation

4.6 REPRESENTING TRUST: INCORPORATING KONFIDI

Trust plays a crucial role in politics. In order to provide effective support for self-organization of people in politics, we need to represent who trusts who and in what subject. Overall trustworthiness with respect to a given subject (i.e. an issue) could then be computed and used as an ordering of recommended people and/or their solution plans. Moreover, with suitable formalism it is possible to propagate trust and find out a trustworthiness of someone unknown, which can be useful in a large and distributed environment of the WWW.

Trust can be defined as “the psychological state comprising (1) expectancy: the trustor expects a specific behavior of the trustee such as providing valid information or effectively performing cooperative actions; (2) belief: the trustor believes that expectancy is true, based on evidence of the trustee’s competence and goodwill; (3) willingness to be vulnerable: the trustor is willing to be vulnerable to that belief in a specific context where the information is used or the actions are applied”. [Huang and Fox, 2006] In our case, the trustor usually expects the trustee to achieve promised objectives of the solution plan (first condition). This expectancy is based on the belief in the trustee’s competence in the context of the issue (second condition). Finally, together with this expectancy and belief, trustor also run the risk that trustee may not behave as expected, i.e. that s/he will behave contrary to the promise (third condition).

Representation of trust on the semantic web has been widely researched and several ontologies have been proposed. Golbeck et al. [2003] pioneered in the development of ontology for representation of trust in social networks. They developed an ontology¹⁴ based on FOAF schema, where trust is represented either generally as a relation between two instances of foaf:Agent, or it is represented as an instance of TopicalTrust class that is related to a trustor, a trustee, a topic and a trust value. For example, Martin trusts Arthur regarding computer programming, but do not trusts him in the context of driving. The ontology also provides a possibility to measure a trust or distrust at a discrete scale from 0 to 10: 0 is for *distrust absolutely*, whereas 10 means *trusts absolutely*. Brondsema and Schamp [2006] developed a system called Konfidi¹⁵ combining a PGP’s web-of-trust¹⁶ with a trust network based on their own schema. The system is intended to serve as a multi-purpose platform for deriving a trustworthiness in a distributed environment. For example, it can be used to overcome

¹⁴ See <http://trust.mindswap.org/trustOnt.shtml>.

¹⁵ The word “konfidi” is the Esperanto term for trust. For more information about the project, see <http://konfidi.org/>.

¹⁶ PGP stands for Pretty Good Privacy. See <http://www.pgpi.org/doc/pgpintro/>.

some of the current pitfalls of spam filters. Similarly to the schema of Goldbeck et al., the trust ontology used in Konfidi (hereafter, *Konfidi schema* or simply *konf* are used) is built on top of FOAF. On the left part of the figure 16, the vocabulary of Konfidi schema is illustrated. Trust is represented as a relationship between two instances of `foaf:Agent` and the relationship is related to at least one trust item (an instance of `konf:Item`), which describes a context of the trust. A rating of the trust item is a real number within the interval $[0, 1]$. A topic of the trust item is an instance of any OWL class. Note that in contrast with the ontology of Goldbeck et al., the term *truster* is used instead of *trustor*. Both *truster* and *trustor* are freely interchangeable in this thesis. Dokoochaki and Matskin [2008] analyzed these two ontologies (beside others) and created their own that is similar to the Konfidi schema. Their schema also represents trust as a relationship, but its parameters are splitted into `AuxiliaryProperties` and `MainProperties`. The instance of the latter class is related to a subject (i.e. a topic) and to a value (i.e. a rating). Both subject and value are data-type properties. Auxiliary properties represents optional parameters of trust like begin and end dates of the trust, its goal, etc.

We decided to prefer a continuous measure to a discrete one, because we believe it is more flexible, while it is still possible to easily transform it to a discrete one whenever needed, so the ontology of Goldberg et al. is not applicable. The ontology of Dokoochaki and Matskin is also unsuitable for our needs, because the subject is represented as a literal, but we need it to be a resource. Another reason for rejection of this ontology is (from our point of view) redundant `AuxiliaryProperties` class. We believe these properties can be directly related to an instance representing a trust item. Optionality of these properties can be formalized as corresponding cardinality restrictions. Taken this into account, we have finally chosen the Konfidi schema, because it uses both continuous scale as a measure of trust and allows us to express a topic of a trust as a resource. In addition, using this ontology brings the possibility of using the Konfidi system for trust computation in the future.

An example of trust representation is depicted in the figure 16. Because the SIOC ontology defines the `sio:Item`, we have created the class `opol:TrustItem` that is a subclass of `konf:Item`, so as to keep the whole schema comprehensible. A range of the `konf:topic` property is an instance of `owl:Thing`¹⁷, but in OPOL it is primarily an instance of either `doldm:promise` or `opol:PoliticalIssue`. This formalism enables a person to declare trust to either another person or a group and in either the whole issue (i.e. without exceptions), or the trust is restricted to the trustee's promise, which means the trustor trusts trustee in a context of a particular role the promise is related to. Declared support by one person to another can be conceived as an implicit trust relation, because it does not make sense to support somebody in politics which is not trustworthy from the supporter's point of view. The right part of the diagram illustrates this fact by explicitly representing trust which is implicitly present in the support relation depicted in the figure 13 on page 31.

¹⁷ `owl:Thing` is a class of all individuals. Therefore every class in OWL is a subclass of this class.

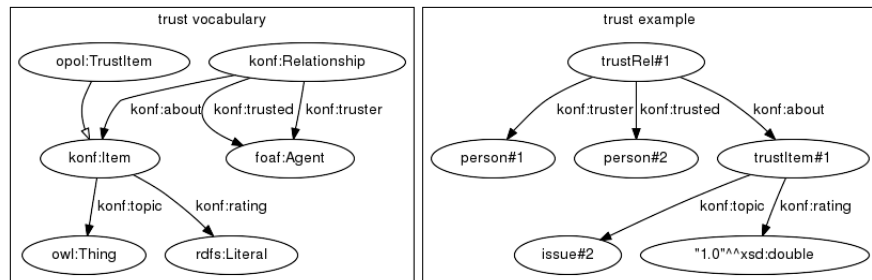


Figure 16: Trust representation in OPOL

Value your freedom or you will lose it, teaches history. "Don't bother us with politics", respond those who don't want to learn.

— Richard Stallman

A proof-of-concept application called Ontopolis.net has been developed as a part of the thesis and we present its architecture here. First, we conclude architectural constraints and requirements which result from previous part describing theoretical foundations of our work. Then we describe the system from the bird's eye perspective and we provide a short introduction to the key technologies the system is built on. We also focus on some implementation aspects like transaction handling, connection management, real-time reasoning, similarity computing, storing and constraint validation of RDF data. Finally, the user interface and typical use cases are described. The testing version of the system is available on-line¹ and the testing data-set mentioned throughout this chapter can be found in project's SVN.²

5.1 ARCHITECTURAL CONSTRAINTS AND REQUIREMENTS

As we have mentioned in the section 2.2 on page 8, the implemented system should not *a priori* conceive any particular political issue nor organization. Moreover, because the system is an environment in which politically engaged persons are self-organizing, the possibility of *free interaction* between users with one another as well as between users and their environment is a must. This implies that users have to have *power to modify the system*, to freely join or withdraw from it and also it is crucial that users have a *freedom of speech and association*. The self-organization means that no one controls the process. We believe that the only possibility how to guarantee these rights and possibilities is to make the whole system as a *free/libre open source software*. Therefore, all project's files are available under GPL v3 license.³

Nevertheless, the system is only an environment and the most important is people themselves: their thoughts, work and content they have created. Nobody should be forced to use a system which s/he don't want to use and so together with guaranteeing the free interaction inside the system, it is also needed that people have the right to choose their environment. We claim that RDF together with OWL schema are a good way how to implement this need, because even if a user decides to withdraw from the system, s/he can easily transfer his/her data into another system. The freedom of a person on the Web is to a great extent determined by a *freedom of his/her data*. Richard

RDF and OWL are described in the section 3.1 on page 13.

¹ See <http://www.ontopolis.net>.

² See http://ontopolis.svn.sourceforge.net/viewvc/ontopolis/trunk/Ontopolis/web-app/WEB-INF/testing_data.xml?view=markup&pathrev=51.

³ Project's homepage is <http://ontopolis.sourceforge.net>. GPL v3 can be found at <http://www.gnu.org/licenses/gpl-3.0-standalone.html>.

Stallman recognized the importance of the right to share, to modify and to co-operate in the development of software⁴, because this model is community-based instead of oligopoly-based and hence much more competitive and de-monopolized, which in turn leads to freedom of users of free software. The social software itself is useless without data and so it is very important to have the right to manipulate them freely as well.

In order to provide an access to the system to as many people as possible, we have decided to implement it as a web application. With respect to the architecture this means that application logic should be *stateless* if possible and that it has to deal with multiple requests at one time, so that underlying database has to provide *transactions*.

Because all data are in RDF, it is suitable to reason over them. However, reasoning over large amount of data can be very demanding of resources. Therefore, we need a real-time reasoning which scales even if the knowledge base is quite large.

5.2 ARCHITECTURE AND IMPLEMENTATION OF ONTOPOLIS.NET

The Ontopolis.net is based on the Grails framework⁵, which is a platform for agile development of WWW application. It is implemented in Groovy language⁶ that is an agile and dynamic language for the Java platform. The Grails itself consists of several other popular Java frameworks that it integrates together using Convention-over-Configuration design pattern.[Smith and Ledbrook, 2009, p. 6] This pattern provides a developer with smart defaults of almost all configuration options of the software architecture and so s/he can effectively focus on the code instead of writing configuration files. On the other hand, if it is necessary to change any configuration option, it can be done easily and most of the time without the use of XML configuration files, which are otherwise ubiquitous in that case in other frameworks like Struts⁷.

The figure 17 is an illustration of the system's architecture from the bird's eye perspective. The application has a 3-tier architecture according to the Model View Controller (MVC) pattern.⁸ User interacts with an interface implemented as a set of interlinked web pages on the server side. User's request is dispatched by the framework to a controller, which calls an appropriate service. Services provide an application logic and serve as a façade for corresponding objects from an ontology. All data is stored at Jena's SDB⁹ RDF store that uses the PostgreSQL database at the backend. We also use PostgreSQL directly for generating unique identifiers for various instances of classes from OPOL (e.g. persons, plans, etc.). Jena's rule engine is used for real-time reasoning over RDF data and Pellet reasoner is used for constraint checking.

⁴ He is a founder of the GNU project and an author of the General Public License (GPL). See <http://www.gnu.org/>.

⁵ See <http://www.grails.org/>.

⁶ See <http://groovy.codehaus.org/>.

⁷ See <http://struts.apache.org/>.

⁸ See <http://java.sun.com/blueprints/patterns/MVC.html>.

⁹ See <http://jena.sourceforge.net/SDB/>.

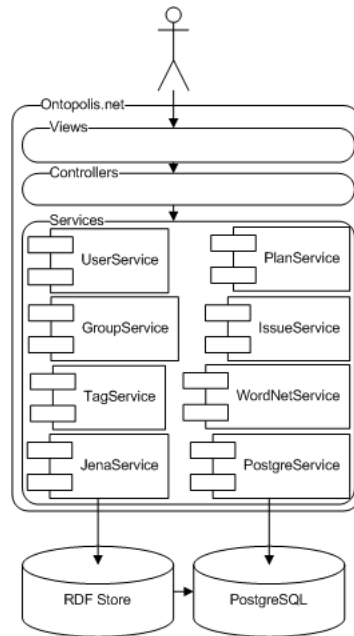


Figure 17: General architecture of Ontopolis.net

5.2.1 Grails framework

Our description of Grails and related technologies is really brief and the aim is not to provide a fully comprehensive description. We are only focusing here on some key aspects of these technologies, which either lead us to the decision to use them, or which are crucial for solution of some of the unique problems arising from aforementioned requirements and constraints.

Groovy programming language

Groovy is a language for the Java platform, which means it is bytecode-compatible with Java language and all others with compilers to Java bytecode. Consequently, every Java library can be directly used in Groovy program and vice versa. It has a Java-like syntax and in fact most of the time it is possible to simply cut-and-paste a Java code into a Groovy source file and it will work. However, Groovy provides many features making the developer's life easier:

- dynamic typing
- scripting support
- native support of collections
- closures
- metaprogramming

It is not necessary to declare a type of a variable in Groovy, because if the type declaration is omitted, it is *dynamically* determined from the context. In spite of the fact Groovy is as well as Java object-oriented language, a programmer is not forced to work with objects every time.

```

1 groovy> def names = ["Arthur", "Martin"]
2 groovy> def lengths = names.collect {name->name.size()}
3 groovy> println lengths
4
5 [6, 6]

```

Figure 18: Closure in Groovy

It is possible to write a simple *script* without definition of a class. This is similar to work with objects and types in other scripting languages like Python.¹⁰ Iterating through collections or arrays in Java is sometimes tough, because first it is necessary to obtain an iterator and then iterate over it in a *for* cycle. Groovy adds additional methods for iterating over collections and arrays that enable to find a specific value, collect a result of a specific operation on each element of a collection, etc. This specific operation is passed to the general-purpose methods via a *closure*, which is actually an implementation of 1st class functions in Groovy.¹¹ The figure 18 provides an example of this feature. First line defines a *names* variable, which is a list of names (i.e. String values). Second line calls a *collect* method with a closure as an argument. This closure is called on each element of the list. Therefore, closure provides a practical way to name a fragment of code, i.e. an algorithm, and work with it as with an ordinary variable. Groovy is not dynamic only because it determines a type of a variable in runtime, but because it supports a modification of classes in runtime and so it is possible to create a completely new method even during the time of its invocation. This feature is called a *metaprogramming* and it is widely used in Grails.

Spring framework

Spring framework¹² integrates all components that Grails consists of. Every component is treated by Spring as a *service* providing an *interface*, which is a set of methods and constants that are provided by the service. Mechanism of integration of services in Spring is inspired by the Dependency Injection pattern.[Fowler, 2004] According to this pattern, when one service is dependent on another one, instead of directly creating an instance of this service, it only declares a variable and the instance of the service is automatically created and then *injected* by the framework to this variable. Therefore, both services are decoupled and more reusable. Moreover, it is possible to completely change the implementation of the service only by changing the configuration as far as the *interface* remains the same.

This puzzle-like architecture of Spring allows it to easily integrate additional modules. In Grails, the Web MVC and Web Flow modules¹³ are used. The former is intended to build MVC web applications, and thus it is capable to dispatch HTTP requests to services, to bind data from HTTP forms to Java classes and to validate them, and to control returned views. The latter is build on top of Web MVC module and

¹⁰ See <http://www.python.org>.

¹¹ See <http://groovy.codehaus.org/Closures>.

¹² See <http://www.springsource.org/about>.

¹³ See <http://www.springsource.org/webflow>.

provides advanced support and control of operations, which typically consists of more steps (e.g. wizards).

GSPs and Sitemesh

Groovy Server Pages ([GSP](#)) is a technology for generating HTML with Groovy. Although it is possible to use traditional Java Server Pages in Grails, using GSPs has some benefits like the Safe Navigation operator¹⁴, which effectively avoids `NullPointerException`. Grails allows to create distinct templates generating markup for e.g. one kind of object and then to re-use this template in many different views.

In addition, the Sitemesh is also integrated into Grails. It is a framework for creating web layouts and decorating views. It helps to create consistent layout of a site with a lot of pages.

Unit and integration testing

Grails is an *agile* framework, and so it provides an out-of-the-box capability of unit and integration testing. We covered all key services in Ontopolis.net by integration tests, because many of features of the system are not typical, and thus the development cycle was quite short and the code was being continuously refactored. On the contrary, we didn't used any unit tests, because we always needed to test the functionality depending on the connection to the RDF store and/or database.

5.2.2 *Use of Jena for working with RDF*

Jena¹⁵ is an open-source semantic web framework. It is actually a whole ecosystem of modules and tools for working with RDF and OWL. The core of the framework consists of a set of interfaces.¹⁶ The key interface in Jena is `Model`, which represents an RDF model that can be stored in a memory, a file or in a relational database. The model is a set of triple statements and it defines methods for manipulating it, i.e. to query, to add, to remove or to reify a statement. Elements of triples are represented by `Resource`, `Literal` and `Property` interfaces.

Storing and querying an RDF graph

Jena supports SPARQL and SPARUL queries by its ARQ¹⁷ query engine. ARQ provides several extension to standard SPARQL. We use *property paths*, *sub-select queries*, *NOT EXISTS*, *GROUP BY* and *HAVING* extensions in Ontopolis.net. Property paths is to graph patterns what regular expressions is to strings. For example, to match all known persons in distance 1 to 2, this property path can be used: `?person foaf:knows{1,2} ?nearPerson`. Sub-select queries, *NOT EXISTS* and *GROUP BY/HAVING* features are similar to their counterparts in SQL.

Ontopolis.net uses RDF as the only datamodel. There are several RDF stores available for Jena. TDB¹⁸ is a non-transactional pure-Java RDF

¹⁴ See <http://groovy.codehaus.org/Operators>.

¹⁵ See <http://jena.sourceforge.net/>.

¹⁶ See JavaDoc: <http://jena.sourceforge.net/javadoc/index.html>.

¹⁷ See <http://openjena.org/ARQ/>.

¹⁸ See <http://openjena.org/TDB/>.

```

1 def model = jenaService.getPlainModel()
2 model.begin()
3 try {
4     model.add(foo,bar,foo)
5     ...
6     model.commit()
7 } catch (JenaException e) {
8     model.abort()
9     log.error(e)
10 } finally {
11     model.closeAll()
12 }

```

Figure 19: Transaction handling example

store intended for large datasets and high performance. RDB¹⁹ or SDB²⁰ are RDF stores with a relational database on backend. Because the TDB is non-transactional and RDB is deprecated, we have chosen SDB. In addition, SDB also supports named graphs that simplifies management of models.

SDB supports all frequently used relational database. Because Ontopolis.net is built as a free software, we have considered only open-source databases. Namely, MySQL²¹ and PostgreSQL²² were possible candidates. Finally, we have chosen PostgreSQL because it has a native support for sequences, which is a mechanism for generating unique sequence of numbers. We need this feature for generating unique identifiers for instances of various classes from OPOL.

Transaction and connection management

Jena's Model provides three methods to support transactions: `begin()`, `commit()` and `abort()`. The figure 19 is an example of typical transaction usage idiom. The model object is obtained first (line 1), new transaction is created (line 2) and after some modifications of RDF graph is made (line 4-5), they are either committed at once (line 6) or wholly rejected if any Jena-related error occurs (line 8). Finally, the model is closed anyway (line 11).

Each user request needs its own database connection which brings the need of database connection pooling. Although Model supports transaction, it is rather a high-level interface and its `close()` method does not close the underlying database connection. In addition, access to RDF statements in Jena SDB is realized though the Store interface, which can be then called directly via SPARQL, or it can be connected to the Model. Neither the Store's `close()` method closes the connection. But the connection should be returned to the pool as soon as possible. We have considered two solutions of this problem. First is to create a special wrapper class with method that closes both the model and the underlying database connection. Second is to use metaprogramming feature of Groovy and dynamically add this method straight to the

¹⁹ See <http://jena.sourceforge.net/DB/index.html>

²⁰ See <http://openjena.org/SDB/>.

²¹ See <http://www.mysql.com/>.

²² See <http://www.postgresql.org/>.

Jena's implementation of `Model`. Because the concrete implementation of `Store` depends on the database engine (i.e. MySQL, PostgreSQL, etc.) and it is chosen by a factory class, the wrapper has to be either loosely-coupled to the implementation of `Store` in the sense it is not its subclass, or there will be a special sub-class wrapper for each of the implementations of `Store`. We find this solution quite inflexible, because the factory class can change in time and the implementations of `Store` as well. Second solution has at least one pitfall, because it disables the compiler to uncover the error of missing method during compiling the source code because the method is added dynamically. However, this still outweighs the pitfalls of the solution based on wrapper class.

The figure 20 depicts two methods in `JenaService` that returns `Store` and `Model` objects with added `closeAll()` method. When the `getStore()` method is called, it first gets a connection to database (line 2) and then registers new method `closeAll()`. When it is called, it first returns the connection back into the pool (line 6) and then closes the store object as well (line 7). The `getPlainModel()` method is very similar to `getStore()` — it first obtains a store object (line 15) and when the `closeAll()` is called, it first propagates this call to the store (line 20) and then closes the model (line 21).

This solution preserves the possibility to change underlying database engine and does not lead to incompatibilities with future versions of Jena as long as the interfaces remain the same.

Connection pooling is provided by Apache DBCP library²³ and it is configured by Spring framework. General access to the pool is provided by `PostgreService`, which also implements a method for getting unique ID numbers from database sequences.

Reasoning

Another important interface in Jena is `Reasoner`, which is implemented by various available reasoners — both internal, i.e. distributed as a part of Jena, or external, i.e. developed by a third party. Jena's reasoning support is based on a general-purpose rule engine that supports three types of chaining: *forward*, *backward* and *hybrid*. If using a forward-chaining rule, all its consequences are added to a model during its initialization. On the contrary, a backward-chaining rule is fired during a query and so its consequences are computed on demand. Both these rule-firing strategies have its pros and cons: whereas the forward-chaining is faster for querying and at the same time more demanding on memory, the backward-chaining is less memory-demanding but also slower during querying. Hybrid strategy is a combination of these two. First, all consequences of forward chaining rules are computed and stored in a memory and then the backward chaining rules are fired during the query time. Therefore, it is possible to first fire all frequently used rules once during initialization and then fire special ones on-demand. Jena provides several reasoners out-of-the-box for RDFS and various flavours of OWL, which are based on this general-purpose rule engine.

We use the rule engine for a real-time reasoning with backward-chaining rules. The advantage of using rules is that when modifying the knowledge base it is not needed to add or to delete the statements

²³ See <http://commons.apache.org/dbcp/>.

```

1 def Store getStore() {
2     def conn = postgresService.getConnection()
3     def store = SDBFactory.connectStore(SDBFactory.createConnection
4         (conn), storeDesc)
5     store.metaClass.invokeMethod = {String name, args ->
6         if (name.equals("closeAll")) {
7             store.getConnection().close()
8             return store.close()
9         } else {
10            return store.metaClass.getMetaMethod(name, args).invoke
11                (delegate, args)
12        }
13    }
14    return store
15 }
16 def Model getPlainModel(String graphName) throws
17     NoSuchMethodException {
18     def store = getStore()
19     def model = SDBFactory.connectNamedModel(store, graphName)
20     model.metaClass.invokeMethod = {String name, args ->
21         if (name.equals("closeAll")) {
22             model.abort()
23             store.closeAll()
24             return model.close()
25         } else {
26             MetaMethod method = model.metaClass.getMetaMethod(name,
27                 args)
28             if (method) {
29                 return method.invoke(delegate, args)
30             } else {
31                 throw new NoSuchMethodException()
32             }
33         }
34     }
35    return model
36 }

```

Figure 20: Connection management in Jena

```

1 [groupMember:
2   (?group foaf:member ?person)
3   <-
4   (?person foaf:holdsAccount ?nick),
5   (?usergroup sioc_c:has_member ?nick),
6   (?group foaf:holdsAccount ?usergroup)]

```

Figure 21: Rule for relating groups to persons

that are inferred, and thus it reduces the possibility of errors. The figure 21 is an example of the rule used in Ontopolis.net.²⁴ It infers that a person is a member of a group (line 2) if s/he holds an account (line 4) that is a member of a usergroup (line 5) which is an account of the group (line 6).

The Model interface is the most general way how to work with a set of statements in Jena. It does not reflect a possible inferencing nor it does contain a support for working with ontologies. InfModel is an interface representing a model that has a bounded reasoner into it. OntModel is then an extension of InfModel containing special methods for working with a model of an ontology like creating instances of classes from the ontology.

5.2.3 Use of Pellet for constraint validation

Pellet²⁵ is an open source OWL DL reasoner. It can be used separately or it can be plugged into another software. It ships also with a native implementation of the Reasoner interface and so it can be used directly in Jena framework. The native interface is much more efficient than a DIG interface that is XML-based.²⁶

In standard architecture of the application that uses a relational database, the consistency of data is to a great extent guaranteed by referential integrity checks (i.e. FOREIGN KEY constraints), uniqueness of values (i.e. UNIQUE constraint) and various cardinality restrictions (e.g. NOT NULL constraint). However, OWL adopts the UNA and OWA, which are suitable for open and distributed world of WWW where knowledge about the domain is rather incomplete, but for the closed and centralized environment of one particular application where the knowledge can be assumed as complete it is error-prone, because it makes data validation more complicated. Sirin and Tao [2009] propose a possible semantics to express integrity constraints in OWL and presents a way to transform integrity constraints into SPARQL queries. They have also implemented a library which uses a Pellet reasoner and which can be used for listing of statements that break any of the following types of constraints:

- **Typing constraints** “require that individuals that participate in a relation should be instances of certain types.” (ibid) For example, the

²⁴ The complete list of rule definitions can be found at <http://ontopolis.svn.sourceforge.net/viewvc/ontopolis/trunk/Ontopolis/web-app/WEB-INF/schema/ontopolis.rules?revision=21&view=markup>. Another conceivable rule, which is not implemented, is a rule that infers the trust between the supporter and the candidate; or a rule that infers the belief of the supporter to the candidate's promise.

²⁵ See <http://clarkparsia.com/pellet/>.

²⁶ See <http://clarkparsia.com/pellet/faq/using-pellet-in-jena/>.

The relation between groups and usergroups is a topic of the sections 4.2 on page 26 and 4.3 on page 27.

OWA and UNA are described in the section 3.1.2 on page 16.

statement `?person foaf:knows ?friend` will cause a violation of this constraint type if the knowledge base does not contain a statement assigning a type `foaf:Agent` to both `?person` and `?friend`. On the contrary, these type assignments will be inferred with OWA.

- **Participation constraint** “require that instances of the corresponding class should have a role assertion” (*ibid*). For example, each instance of the `doledns:plan` class has to have its goal (i.e. an instance of the `doledns:goal` class) specified. This is similar to NOT NULL combined with FOREIGN KEY constraint in relational databases.
- **Uniqueness constraint** “require that an individual cannot participate in multiple role assertions with the same role” (*ibid*). It is useful to assume that all persons in Ontopolis.net hold exactly one account. This can be expressed in OWL with a `FunctionalProperty`, but in non-UNA interpretation if a person holds two distinct user accounts, it is inferred that these two accounts are the same. On the other hand, strict UNA interpretation will prohibit to have two different identifiers for the same resource. Authors thus adopts a weak UNA in which this constraint is violated only if these two resources are not inferred via `owl:sameAs` as the same. This constraint is an analogy of FOREIGN KEY combined with UNIQUE constraint in relational databases.

These constraints expressed in OWL are then read by the library and transformed to SPARQL queries that selects all statements violating the constraints. Even though this is not an ideal solution, because errors are not detected in real-time and the whole knowledge base has to be read into memory, it is better than no validation at all. Note that OWL 2²⁷ is more expressive in terms of constraints. For example, it introduces keys that can be used in the similar manner as the FOREIGN KEY constraint.

5.2.4 Programming with ontology

When user sends a HTML form, the data is bound into a *Command Object* which is an ordinary Plain Old Groovy Object (POGO) with constraints static data property that contains validation constraints. [Smith and Ledbrook, 2009, p. 141] Validation of the command object is done after data binding automatically by Grails. If no constraint is violated, the object is saved into the RDF store. Because RDF datamodel is intrinsically different to object-oriented model of Groovy, we have implemented several utility methods for object-to-RDF mapping in `JenaService`. Each command object representing an object from OPOL (e.g. a person, a user or a plan) contains method `getOntologyMapping()` which returns a mapping of class’s field names to Property instances.²⁸ For example, surname field of class `FoafPersonCommand` is mapped to the instance of Property representing `foaf:surname`. This way the fields’ values stored as a literal can be easily saved into the RDF store by a generally-implemented `JenaService.save(co)` method. If the field is not stored as a literal, special methods are implemented in either `JenaService` or special services like `PlanService` or `IssueService`.

²⁷ See <http://www.w3.org/TR/owl2-overview/>.

²⁸ That is to say, the instances of Jena’s Property interface.

Command objects in Ontopolis also implements `getObjectResourceUri()` method that is overloaded by the implementation which accepts one argument representing the identifier of the object. For example, the `FoafPersonCommand` implements it as follows:

```
def static getObjectResourceUri(long id) {
    return "http://www.ontopolis.net/resource/person#{id}".
        toString()
}
```

There are three namespace prefixes in Ontopolis:

1. For OPOL, we use `http://www.ontopolis.net/ontopolis/0.1/` prefix, where the last part is reserved for version of the schema.
2. Resources created in Ontopolis.net have namespace `http://www.ontopolis.net/resource/` followed by a type of the resource and its ID. For instance, user with account name arthur is represented as a resource with URI `http://www.ontopolis.net/resource/user#arthur`.
3. Named graphs have `http://www.ontopolis.net/graph/` prefix. Currently, three named graphs are used. In the first the WordNet is stored, the second contains similarity relations (see next section) and all other data are stored in the third graph.

The results of data manipulation methods in services are tested using integration tests. After data from a command object are stored, it is necessary to check whether the data are stored wholly and whether any constraints defined in OPOL are not violated. Consequently, we use OPOL as a vocabulary as well as a constraint definition. In `JenaService`, we have implemented a method called `printICViolations()`, which validates the knowledge base and prints the violations to the standard output stream. This method is based on library described in section 5.2.3. Because it needs to read the whole knowledge base into a memory we currently not use validation in real time, but we only validate constraints during integration tests so as to uncover mistakes in the data manipulation methods.

5.2.5 Similarity algorithms

Tag disambiguation

When the tagged item is about to be saved into the RDF store, tags are disambiguated by method `TagService.disambiguateTags(tags)`, which accepts a set of tags as an argument and returns a mapping from tags to their most similar tag. The algorithm computes similarities of pairwise combinations of tags and the highest similarity determines the choice of the synset for the given tag. The disambiguation information is then saved into the RDF store as a reification of the statement relating the tagged item to the tag.

Item similarity

Since some tags describing items are disambiguated, the overall similarity between two tagged items can be computed. General-purpose method for computation of similarity of two tagged resources is implemented in `TagService.getTaggedItemSimilarity()` method which

The vocabulary for representation of disambiguation and similarity is discussed in the section 4.5 on page 32.

Similarity measure used in Ontopolis is described in the section 3.3 on page 20.

Algorithm 5.1 The tagged item similarity

1. Let the $tags_1$ and $tags_2$ to be a set of tags of one and another item, respectively.
 2. Let the ct_1 and ct_2 to be a count of $tags_1$ and $tags_2$, respectively.
 3. If the ct_1 is lower than ct_2 , swap $tags_1$ and $tags_2$ as well as ct_1 and ct_2 .
 4. Let the os (i.e. overall similarity) to be a real number initialized with 0.
 5. For each tag_1 in $tags_1$:
 - a) Let the sim to be an empty list of similarities between tag_1 and $tags_2$.
 - b) For each tag_2 in $tags_2$:
 - i. If tag_1 and tag_2 are disambiguated, compute their sim_{lin} and append it to sim ;
 - ii. else if tag_1 equals tag_2 , add 1 to sim ;
 - iii. else add 0 to sim .
 - c) Add maximum of sim to os .
 6. Return $\frac{os * ct_2}{ct_1}$.
-

uses the algorithm 5.1. This measure is symmetric and a returned value lays within the interval $[0, 1]$. Although the similarity of two identical set of tags is 1, this measure can be much lower even if these sets differs only in one tag, because it is their synsets which is taken into account and not their lexical forms. Only when at least one synset of compared tags is not known, the lexical form is taken into account. This enables the system to relate two items which are tagged by the same tags but that are not contained in WordNet. Because the maximum of similarities of pair wise combinations between sets of tags is added to the overall similarity, the outer cycle is always the set with higher count so as to search larger combinatorial space as well as guarantee symmetricity of the measure.

Let us demonstrate the algorithm on the following example, where one plan is described by $tags_1 = \{ "pickpocket", "Praha" \}$ and the second one is described by $tags_2 = \{ "criminality", "Prague" \}$. Because the count of both sets of tags are the same, the third step is omitted. The algorithm then continues in determining the overall similarity as follows:

1. Determine similarities between $pickpocket$ and $tags_2$:
 - a) The similarity between $pickpocket$ and $criminality$ is 0
 - b) The similarity between $pickpocket$ and $Prague$ is 0.059
 - c) Maximal similarity for $pickpocket$ is 0.059
 - d) $os = 0 + 0.059$
2. Determine similarities between $Praha$ and $tags_2$:
 - a) The similarity between $Praha$ and $criminality$ is 0

The appendix B on page 73 contains the results of testing this algorithm on the test data-set.

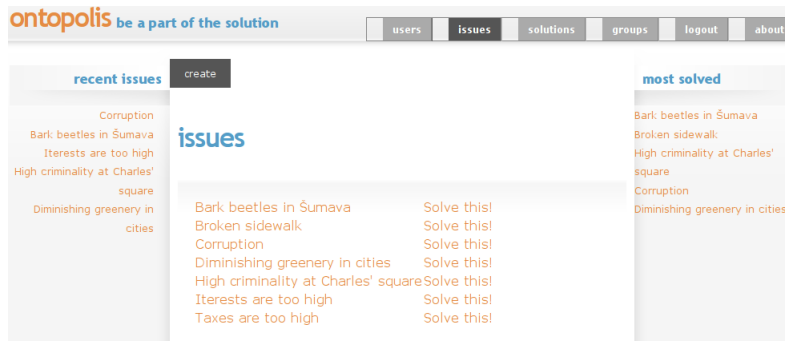


Figure 22: The layout of Ontopolis.net

- b) The similarity between Praha and Prague is 1
 c) Maximal similarity for Praha is 1
 d) $os = 0.059 + 1$
3. The similarity between the plans is $\frac{1.059 \cdot 2}{4} \doteq 0.53$

Determining, storing and querying similar items

After the tagged item is saved into the store, its similarity to other items is computed and stored. The system first determines a set of possible similar items and then computes their similarity measure. The items with similarity above a certain threshold²⁹ are considered as similar and the corresponding similarity relations are saved. These related items are then selected from the store and presented to the user together with the item. For example, the set of plans possible similar to a given plan consists of another plans associated by a joint tag, synset and/or issue. The more of these items are shared the higher is the chance of the associated plan to be treated as similar to the given plan. The selection of possible similar items is typically limited to maximally 5 items, because the computation of similarity of items in testing dataset typically lasted for about 400ms.³⁰ The complete SPARQL query for selection of possible similar plans is presented in the section ?? in the appendix C on page 77, in which other queries for selection of possible similar issues and for similar users are also listed. Users are not tagged and so only the facts like joint membership determines the similarity of users, though.

5.2.6 *User interface*

Each page in the system has a three-column layout as it is depicted in the figure 22. The middle column provides main information of the page and the side columns provide compact information boxes with related content. For example, during the creation of a new object (i.e. a plan, an issue, etc.), the list of recently added items is displayed in order to prevent creation of duplicates. A visitor of Ontopolis.net is provided with answers to some of the competency questions as early as on the frontpage of the site. S/he sees what are actual political issues and solutions; which solution plan was added recently and who is the

The competency questions are listed in the section 4 on page 25.

²⁹ The exact value of the threshold is a matter of experimentation and we use 0.3 currently.

³⁰ On PC with AMD Phenom II 2800Mhz CPU and 4GB RAM.

most trusted or most active person in the system. The last two kinds of information are very important in every social site, because they motivates users to be active. The user's activity is measured by the count of issues, plans or groups s/he created. The most trusted person list is computed by ordering users by the count of their support.

When the user logs in, s/he sees the user's details page with his/her profile, friends and people s/he knows. The difference between the last two is that friends are *symmetric* foaf:knows relation whereas s/he can know a person for whom s/he is unknown. In another words, known people are those ones related with an *asymmetric* foaf:knows relation. Further, the user's details page also shows the groups the user is a member of and his/her goals.

The Issues section provides a list of all issues and lists of the most recent and the mostly solved issues. In the detail page of the issue, it is possible to start a wizard for creation of a solution plan for the issue. In the wizard, the user specifies any additional issues s/he wants to solve, the goal of the plan, its measures and finally s/he describes the plan as a whole. Contextual help is provided in the side information box during this process. At the end of the wizard, the plan's detail page is displayed that shows who is the author of the plan, who are followers, what issues this plan may solve and what are the similar plans. The user can also lists all plans, which are then sorted by a count of the support that the author of each particular plan has. This support can be declared at the plan's detail page. This is the only mechanism to declare a trust to a user in Ontopolis.net presently.

Any user can also create a group and then s/he tacitly becomes its administrator who has the permission to add a plan to the group. Only the plan which the administrator is an author of is allowed to add. It means that it is not possible to "steal" a plan of someone else. Currently, only the founder of the group can be its administrator. After the plan is adopted by the group, this group plays a role of political candidate in the plan and it is possible for other members to adopt it as well. Therefore creation of the group is the only way to share a plan with others. The group's detail page then contains information about the goals of the group, its members and their roles, which is one of a candidate, a follower or no role at all. This typology of roles follows the idea that whereas one person only wants to declare his/her belonging to a group and so s/he becomes a member without a role; another one wants to declare his/her support to a particular candidate and plan and s/he consequently becomes a follower and finally another one publicly declares the promise to implement the plan and hence s/he becomes a candidate.

CONCLUSION AND FUTURE WORK

Present democratic system is based on the competition of political parties, but this competition is biased by the personal relationships between political parties' secretaries and elite politicians, the high cost of entry, and last but not least by the fact that the system of government is based on relatively obsolete technologies based on a static form of writing. Information and knowledge technologies are usually not used to overcome these limits, but rather to scaffold them. But we believe that these technologies have a bigger potential to foster democratic culture. Namely, they can be used for larger participation of the electorate, to re-use effectively its knowledge and to help it to make more mindful decisions. Since the widespread of the Internet, several e-participation, e-democracy and collaborative decision-making systems have been introduced. However, they are either merely a barometer of public opinion (i.e. they are *reactive*) or they are *active* indeed but at the same time out of the scope of the constitutional framework.

We have presented a way to self-organize politically engaged groups of citizens, which we believe may effectively overcome this antagonism of present e-participation systems. First, we have created the *core ontology* for description of political programs, commitments and trust between people, as well as their mutual relationship. This ontology can also help other existing systems to enhance their mutual interoperability. Second, we have developed new and accommodated existing *algorithms* for support of self-organization of people. Namely, algorithms for computing of semantic similarity between program descriptions and for enhanced social recommendation were researched. Finally, we have implemented a proof-of-concept social-semantic web *application*.

The self-organization process in Ontopolis.net is *backed* by the core ontology. This enables the system to provide the users with better recommendation of similar-minded companions. Hence the self-organization is based on shared goals and it leads to emergence of authorities in a particular problem domain. The crucial requirements of this process are the rights of citizens, which are guaranteed by the fact the system as a whole is a *free/libre software*. Moreover, the data created by the users are freely portable, because they are represented in RDF. We claim that only both free software as well as free data fulfill the requirements of a self-organizing system. This design enables the competition between various systems and guarantees the right of choice of the system to each user.

The design and implementation of Ontopolis.net has been briefly described and the whole project is available under the terms of GPL v3 on-line.¹ Various particular problems of implementation of a social-semantic web application has been discussed. Namely, the transaction and connection management, the constraints validation of data in RDF, the real-time reasoning and the computation of semantic similarity have been described.

¹ See <http://ontopolis.sourceforge.net>.

The thesis will hopefully also contribute to a discussion of the role of a citizen in democracy, because it uncovers a potential of the presently available technologies to enable the citizen to be a part of the solution of the problem instead of to be a part of the problem itself.

FUTURE WORK

The systems described in the section 1.3 on page 4 are forum and/or wiki-based systems, therefore they have typical problems caused by organizing participants' contributions by time rather than logic.[Klein, 2007] Klein and Iandoli [2008] describe the Collaboratorium system, which was used to structure Italian students' argumentation about energetic issues. Argumentation is structured in this system in the form of an argumentation map, which consists of *issues* (questions to be answered), *options* (alternative answers for a question), and *arguments* (claims that support or detract some other statement).[Klein, 2007] Such a map can be formalized by a directed graph where nodes are issues, options and arguments, while an option cannot be a parent of an issue and an argument cannot be a parent of neither an issue nor an option. Edges connecting arguments to options can be typed describing a fact that argument is based e.g. on an analogy or an authority. This logical organization of users' contributions could lead to better solution in terms of considering more options and choosing the most appropriate of them as well as keeping focus on the main issue. According to Klein, argumentation maps can overcome many of the concerns of other tools used for collective problem solving like blogs, group decision support systems or prediction markets. In future, some argumentation ontology will be used in Ontopolis.net to organize users' arguments.

The users will also be able to declare trust explicitly and the emergent trust network in Ontopolis.net can be used in the future by some advanced democratic decision-making algorithms as described in the section 2.4 on page 10. This will provide the users with the possibility of collective decision making, and thus, for example, they will be able to collaboratively decide the goals of the group, which are now managed solely by the group administrator.

We believe that aligning the whole set of ontologies imported in OPOL with DOLCE would also be fruitful, because it will make the underlying assumptions of all defined concepts more clear and it will also bring better interoperability with other schemas.

The constraint validation process may be better supported using the newly introduced features of OWL 2.

As Groovy is dynamic and hence sometimes much slower than Java, some critical parts of Ontopolis.net with respect to performance will be rewritten into Java.

In future, the REST interface will provide an access to RDF data in the system and some role-based access control mechanism has to be implemented in order to provide this functionality in a safe manner. The Grails framework integrates the Spring Security² that provides this functionality. In addition, this module also provides an integration with OpenID³, which is another desired feature to be implemented.

² See <http://static.springsource.org/spring-security/site/>.

³ See <http://openid.net/>.

Since we currently use the slightly modified version of Java WordNet Similarity Library⁴ that uses the file representation of WordNet for computing the sim_{lin} , one of the advisable extensions is to re-implement this library to use the RDF representation of WordNet in order to not to have two copies of the same data in the application.

The most important is to test the application in a real-world use-case. Several implementation details has to be finalized in order to enable this. First, all result sets have to be paginated. The pagination has to be implemented only in the view layer, because it has been implemented in the underlying services already. Second is to implement the hierarchies of issues and third is to allow update operations on items in the system.

⁴ See <http://nlp.shef.ac.uk/result/software.html>.

BIBLIOGRAPHY

- Grigoris Antoniou and Frank van Harmelen. *The Semantic Web Primer*. MIT Press, 2nd edition, 2008. ISBN 978-0-262-01242-3. original. (Cited on pages 16 and 59.)
- Albert-László Barabási. *Linked*. Paseka, 2005. ISBN 80-7185-751-3. Czech translation: V pavučině sítí. (Cited on page 9.)
- David Brondsema and Andrew Schamp. Konfidi: Trust networks using pgp and rdf. In *Proceedings of the WWW'06 Workshop on Models of Trust for the Web (MTW'06)*, 2006. URL <http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-190/paper01.pdf>. (Cited on page 34.)
- CSB. Czech chamber of deputies election 2006 - part 2, 12 2006. URL [http://www.czso.cz/csu/2006edicniplan.nsf/t/5100356482/\\$File/4205068.pdf](http://www.czso.cz/csu/2006edicniplan.nsf/t/5100356482/$File/4205068.pdf). Czech Statistical Bureau report. (Cited on page 3.)
- DCTerms. Dcmi metadata terms. <http://dublincore.org/documents/dcmi-terms/>, 1 2008. (Cited on page 26.)
- Nima Dokoohaki and Mihhail Matskin. Effective design of trust ontologies for improvement in the structure of socio-semantic trust networks. *International Journal On Advances in Intelligent Systems*, 1(1942-2679):23–42, 2008. URL http://www.ariajournals.org/intelligent_systems/intsys_v1_n1_2008_paged.pdf. (Cited on page 35.)
- DOLH. Laboratory for applied ontology - dolce. <http://www.loa-cnr.it/DOLCE.html>, 11 2009. (Cited on page 29.)
- FOAF. The friend of a friend (foaf) rdf vocabulary. <http://xmlns.com/foaf/spec/>, 11 2007. (Cited on page 26.)
- Martin Fowler. Inversion of control containers and the dependency injection pattern. <http://martinfowler.com/articles/injection.html#InversionOfControl>, 1 2004. (Cited on page 40.)
- FSF. What is free software and why is it so important for society? <http://www.fsf.org/about/what-is-free-software>, 11 2009. (Cited on page 59.)
- Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with dolce. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web. (EKAW '02)*, pages 166–181, London, UK, 2002. Springer-Verlag. ISBN 3-540-44268-5. (Cited on page 29.)
- Aldo Gangemi, Stefano Borgo, Carola Catenacci, and Jos Lehmann. Task taxonomies for knowledge content. <http://metokis.salzburgresearch.at/files/deliverables/>

- [metokis_d07_task_taxonomies_final.pdf](#), 6 2004. (Cited on page 29.)
- Carlos Gershenson and Francis Heylighen. When can we call a system self-organizing? In *ECAL*, volume 2801, pages 606–614, 2003. URL <http://springerlink.metapress.com/openurl.asp?genre=article&iissn=0302-9743&volume=2801&spage=606>. (Cited on pages 8 and 12.)
- Jennifer Golbeck, Bijan Parsia, and James Hendler. Trust networks on the semantic web. In *Proceedings of Cooperative Intelligent Agents*, pages 238–249, 2003. (Cited on page 34.)
- Michael Gruninger and Mark S. Fox. The role of competency questions in enterprise engineering. In *IFIP WG5.7 Workshop on Benchmarking - Theory and Practice, Trondheim, Norway*, 1994. URL <http://www.eil.utoronto.ca/public/benchIFIP94.ps>. (Cited on page 25.)
- Francis Heylighen. Collective intelligence and its implementation on the web: algorithms to develop a collective mental map. <http://pespmc1.vub.ac.be/Papers/CollectiveWebIntelligence.pdf>, 1999. (Cited on page 10.)
- Francis Heylighen. *Encyclopedia of Library and Information Sciences*, chapter Complexity and Self-organization. Taylor & Francis, 2008. URL <http://pespmc1.vub.ac.be/papers/elis-complexity.pdf>. (Cited on pages 7, 9, and 59.)
- Jingwei Huang and Mark S. Fox. An ontology of trust: formal semantics and transitivity. In *Proceedings of the 8th international conference on Electronic commerce (ICEC '06)*, pages 259–270, New York, NY, USA, 2006. ACM. ISBN 1-59593-392-1. doi: <http://doi.acm.org/10.1145/1151454.1151499>. (Cited on pages 34 and 59.)
- Mark Klein. Achieving collective intelligence via large-scale on-line argumentation. *Internet and Web Applications and Services, International Conference on*, 0:58, 2007. doi: <http://doi.ieeecomputersociety.org/10.1109/ICIW.2007.13>. URL <http://ssrn.com/abstract=1040881>. (Cited on page 52.)
- Mark Klein and Luka Iandoli. Can we exploit collective intelligence for collaborative deliberation? the case of the climate change collaboratorium. *MIT Sloan School of Management Working Paper*, 4675, 2008. (Cited on page 52.)
- Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8. (Cited on page 20.)
- LoCY. Localocracy. <http://www.localocracy.org>, 2009. (Cited on page 4.)
- Pierre Lévy. *Collective Intelligence*. Perseus Books, 1999. ISBN 0-7382-0261-4. (Cited on pages 10, 11, 12, and 59.)

- Thomas W. Malone, Robert Laubacher, and Chrysanthos Dellarocas. *Harnessing crowds: Mapping the genome of collective intelligence. MIT Sloan School of Management Working Paper, 2*, 2009. (Cited on page 10.)
- Peter Mika. *The Semantic Web – ISWC 2005*, chapter Ontologies Are Us: A Unified Model of Social Networks and Semantics, pages 522–536. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-29754-3. doi: 10.1007/11574620. (Cited on page 32.)
- George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, 1995. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/219717.219748>. (Cited on page 19.)
- Karel B. Müller. *Political Sociology*. Portál, 2008. ISBN 978-80-7367-380-2. Original in Czech: Politická sociologie. (Cited on page 4.)
- OSS. Open source initiative. <http://www.opensource.org/>, 12 2009. (Cited on page 59.)
- Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. <http://www.w3.org/TR/rdf-sparql-query/>, 1 2008. (Cited on page 18.)
- C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979. ISBN 978-0408709293. URL <http://www.dcs.gla.ac.uk/Keith/Preface.html>. (Cited on page 20.)
- Marko A. Rodriguez and Jennifer H. Watkins. Revisiting the age of enlightenment from a collective decision making systems perspective. *CoRR*, abs/0901.3929, 2009. (Cited on pages 10, 11, and 12.)
- Marko A. Rodriguez, Daniel J. Steinbock, Jennifer H. Watkins, Carlos Gershenson, Johan Bollen, Victor Grey, and Brad deGraf. Smartocracy: Social networks for collective decision making. *Hawaii International Conference on System Sciences*, 0:90b, 2007. ISSN 1530-1605. doi: <http://doi.ieeecomputersociety.org/10.1109/HICSS.2007.484>. URL http://public.lanl.gov/jhw/Publications_files/smartocracy.pdf. (Cited on page 11.)
- Andy Seaborne and Geetha Manjunath. Sparql/update - a language for updating rdf graphs, 4 2008. URL <http://jena.hpl.hp.com/~afs/SPARQL-Update.html>. (Cited on page 19.)
- SIOCC. Sioc core ontology specification. <http://rdfs.org/sioc/spec/>, 1 2009. (Cited on pages xvi, 27, and 28.)
- Evren Sirin and Jiao Tao. Towards integrity constraints in owl. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, 10 2009. URL http://ceur-ws.org/Vol-529/owlled2009_submission_35.pdf. (Cited on pages 17 and 45.)
- Glen Smith and Peter Ledbrook. *Grails In Action*. Manning Publications Co., 2009. ISBN 978-1-933988-93-1. (Cited on pages 38 and 46.)

Wouter van Atteveldt. *Semantic Network Analysis: Techniques for Extracting, Representing, and Querying Media Content*. PhD thesis, Vrije Universiteit Amsterdam, 2008. URL http://vanatteveldt.com/dissertation/vanatteveldt_semanticnetworkanalysis.pdf. (Cited on pages 5 and 30.)

WNOL. Wordnet 3.0 on-line interface. <http://wordnetweb.princeton.edu/perl/webwn>, 12 2009. (Cited on page 59.)

DICTIONARY OF TERMS

- agile development** a set of software development methods based on rapid prototyping, intensive automatized testing and continuous integration (*author*)
- collective intelligence** a form of *universally distributed intelligence*, constantly enhanced, coordinated in real time, and resulting in the effective mobilization of skills[Lévy, 1999, p. 13]
- framework** a set of libraries, utilities and/or tools simplifying the development of software (*author*)
- free/libre software** software that gives you the user the freedom to share, study and modify it. We call this free software because the user is free.[FSF]
- integration test** a test of more units at the same time in order to verify if they work together as they are supposed to work
- ontology** an explicit, formal conceptualization of a domain model[Antoniou and van Harmelen, 2008, p. 114]
- open source software (OSS)** a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.[OSS]
- self-organization** spontaneous emergence of global structure out of local interactions[Heylighen, 2008]
- semantics** the meaning of a word, phrase, sentence, or text[WNOL]
- thesaurus** synonym finder (a book containing a classified list of synonyms)[WNOL]
- transaction** a set of data operations that are either completed wholly (i.e. committed) or wholly rejected (i.e. aborted) (*author*)
- trust** the psychological state comprising (1) expectancy: the trustor expects a specific behavior of the trustee such as providing valid information or effectively performing cooperative actions; (2) belief: the trustor believes that expectancy is true, based on evidence of the trustee's competence and goodwill; (3) willingness to be vulnerable: the trustor is willing to be vulnerable to that belief in a specific context where the information is used or the actions are applied[Huang and Fox, 2006]
- unit test** a test of a *unit*, i.e. a piece of code, that verifies if it does what it is supposed to do (*author*)

Part III

APPENDIX



ONTOPOLIS SCHEMA

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
  <!ENTITY ns "http://rdfs.org/sioc/ns#" >
  <!ENTITY foaf "http://xmlns.com/foaf/0.1/" >
  <!ENTITY access "http://rdfs.org/sioc/access#" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
  <!ENTITY skos "http://www.w3.org/2004/02/skos/core#" >
  <!ENTITY trust "http://www.konfidi.org/ns/trust/1.3#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY ontopolis "http://www.ontopolis.net/ontopolis/0.1/" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <!ENTITY Plans "http://www.loa-cnr.it/ontologies/Plans.owl#" >
  <!ENTITY DOLCE-Lite "http://www.loa-cnr.it/ontologies/DOLCE-
    Lite.owl#" >
  <!ENTITY ExtendedDnS "http://www.loa-cnr.it/ontologies/
    ExtendedDnS.owl#" >
  <!ENTITY ModalDescriptions "http://www.loa-cnr.it/ontologies/
    ModalDescriptions.owl#" >
]>

<rdf:RDF xmlns="http://www.ontopolis.net/ontopolis/0.1/"
  xml:base="http://www.ontopolis.net/ontopolis/0.1/"
  xmlns:DOLCE-Lite="http://www.loa-cnr.it/ontologies/DOLCE-Lite.
    owl#"
  xmlns:ns="http://rdfs.org/sioc/ns#"
  xmlns:Plans="http://www.loa-cnr.it/ontologies/Plans.owl#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:trust="http://www.konfidi.org/ns/trust/1.3#"
  xmlns:access="http://rdfs.org/sioc/access#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ModalDescriptions="http://www.loa-cnr.it/ontologies/
    ModalDescriptions.owl#"
  xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ontopolis="http://www.ontopolis.net/ontopolis/0.1/"
  xmlns:ExtendedDnS="http://www.loa-cnr.it/ontologies/
    ExtendedDnS.owl#"
  xmlns:skos="http://www.w3.org/2004/02/skos/core#">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="file:///home/vasek/Dokumenty/
      skola/diplomka/data/FOAF/foaf-dl.owl"/>
    <owl:imports rdf:resource="file:/home/vasek/Dokumenty/skola
      /diplomka/data/Konfidi/trust.owl"/>
    <owl:imports rdf:resource="http://rdfs.org/sioc/access"/>
```

```

<owl:imports rdf:resource="http://rdfs.org/sioc/ns"/>
<owl:imports rdf:resource="http://rdfs.org/sioc/types"/>
<owl:imports rdf:resource="http://www.loa-cnr.it/ontologies
  /ModalDescriptions.owl"/>
<owl:imports rdf:resource="http://www.loa-cnr.it/ontologies
  /Plans.owl"/>
</owl:Ontology>

```

```

<!--
////////////////////////////////////
//
// Object Properties
//
////////////////////////////////////

-->

```

```

<!-- http://rdfs.org/sioc/ns#has_creator -->

```

```

<owl:ObjectProperty rdf:about="&ns;has_creator">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

```

```

<!-- http://rdfs.org/sioc/ns#has_scope -->

```

```

<owl:ObjectProperty rdf:about="&ns;has_scope">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

```

```

<!-- http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#proper-
part -->

```

```

<owl:ObjectProperty rdf:about="&DOLCE-Lite;proper-part"/>

```

```

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#
intensionally-referenced-by -->

```

```

<owl:ObjectProperty rdf:about="&ExtendedDnS;intensionally-
referenced-by"/>

```

```

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#played-by
-->

```

```

<owl:ObjectProperty rdf:about="&ExtendedDnS;played-by">

```

```

    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<!-- http://www.loa-cnr.it/ontologies/Plans.owl#has-assignment
-->

<owl:ObjectProperty rdf:about="&Plans;has-assignment">
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:ObjectProperty>

<!-- http://www.ontopolis.net/ontopolis/0.1/believes -->

<owl:ObjectProperty rdf:about="believes">
  <rdfs:range rdf:resource="&rdfs;Class"/>
  <rdfs:domain rdf:resource="&foaf;Agent"/>
</owl:ObjectProperty>

<!-- http://www.ontopolis.net/ontopolis/0.1/dissatisfies -->

<owl:ObjectProperty rdf:about="dissatisfies">
  <rdfs:range rdf:resource="&ExtendedDnS;description"/>
  <rdfs:subPropertyOf rdf:resource="&ExtendedDnS;
    intensionally-referenced-by"/>
  <rdfs:domain rdf:resource="&ExtendedDnS;situation"/>
</owl:ObjectProperty>

<!-- http://www.ontopolis.net/ontopolis/0.1/is_solution_of -->

<owl:ObjectProperty rdf:about="is_solution_of">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:domain rdf:resource="&ExtendedDnS;goal"/>
  <rdfs:range rdf:resource="PoliticalIssue"/>
</owl:ObjectProperty>

<!-- http://www.ontopolis.net/ontopolis/0.1/subissue_of -->

<owl:ObjectProperty rdf:about="subissue_of">
  <rdfs:subPropertyOf rdf:resource="&skos;broader"/>
</owl:ObjectProperty>

<!-- http://www.ontopolis.net/ontopolis/0.1/superissue_of -->

<owl:ObjectProperty rdf:about="superissue_of">
  <owl:inverseOf rdf:resource="subissue_of"/>
  <rdfs:subPropertyOf rdf:resource="&skos;narrower"/>
</owl:ObjectProperty>

```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/support_of -->
```

```
<owl:ObjectProperty rdf:about="support_of">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdfs:range rdf:resource="PoliticalRole"/>
  <rdfs:domain rdf:resource="Support"/>
</owl:ObjectProperty>
```

```
<!-- http://www.w3.org/2004/02/skos/core#broader -->
```

```
<owl:ObjectProperty rdf:about="&skos;broader"/>
```

```
<!-- http://www.w3.org/2004/02/skos/core#narrower -->
```

```
<owl:ObjectProperty rdf:about="&skos;narrower"/>
```

```
<!-- http://xmlns.com/foaf/0.1/holdsAccount -->
```

```
<owl:ObjectProperty rdf:about="&foaf;holdsAccount">
  <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
</owl:ObjectProperty>
```

```
<!--
```

```
////////////////////////////////////
```

```
//
```

```
// Data properties
```

```
//
```

```
////////////////////////////////////
```

```
-->
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/password -->
```

```
<owl:DatatypeProperty rdf:about="password">
  <rdfs:domain rdf:resource="&ns;User"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/similarity -->
```

```

<owl:DatatypeProperty rdf:about="similarity">
  <rdfs:domain rdf:resource="SimilarityInfo"/>
  <rdfs:range rdf:resource="xsd:double"/>
</owl:DatatypeProperty>

```

```

<!--
////////////////////////////////////
//
// Classes
//
////////////////////////////////////
-->

```

```

<!-- http://rdfs.org/sioc/access#Permission -->

```

```

<owl:Class rdf:about="&access;Permission"/>

```

```

<!-- http://rdfs.org/sioc/ns#Role -->

```

```

<owl:Class rdf:about="&ns;Role"/>

```

```

<!-- http://rdfs.org/sioc/ns#User -->

```

```

<owl:Class rdf:about="&ns;User"/>

```

```

<!-- http://rdfs.org/sioc/ns#Usergroup -->

```

```

<owl:Class rdf:about="&ns;Usergroup"/>

```

```

<!-- http://www.konfidi.org/ns/trust/1.3#Item -->

```

```

<owl:Class rdf:about="&trust;Item"/>

```

```

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#agent-
driven-role -->

```

```

<owl:Class rdf:about="&ExtendedDnS;agent-driven-role"/>

```

```

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#
description -->

<owl:Class rdf:about="&ExtendedDnS;description"/>

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#goal -->

<owl:Class rdf:about="&ExtendedDnS;goal"/>

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#plan -->

<owl:Class rdf:about="&ExtendedDnS;plan"/>

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#situation
-->

<owl:Class rdf:about="&ExtendedDnS;situation"/>

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#social-
description -->

<owl:Class rdf:about="&ExtendedDnS;social-description"/>

<!-- http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#task -->

<owl:Class rdf:about="&ExtendedDnS;task"/>

<!-- http://www.loa-cnr.it/ontologies/ModalDescriptions.owl#
commitment -->

<owl:Class rdf:about="&ModalDescriptions;commitment"/>

<!-- http://www.ontopolis.net/ontopolis/0.1/AddPlanPermission
-->

<owl:Class rdf:about="AddPlanPermission">
  <rdfs:subClassOf rdf:resource="&access;Permission"/>
</owl:Class>

<!-- http://www.ontopolis.net/ontopolis/0.1/
DeletePlanPermission -->

```

```
<owl:Class rdf:about="DeletePlanPermission">
  <rdfs:subClassOf rdf:resource="&access;Permission"/>
</owl:Class>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/Following -->
```

```
<owl:Class rdf:about="Following">
  <rdfs:subClassOf rdf:resource="&ModalDescriptions;
    commitment"/>
</owl:Class>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/GroupAdmin -->
```

```
<owl:Class rdf:about="GroupAdmin">
  <rdfs:subClassOf rdf:resource="&ns;Role"/>
</owl:Class>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/IssueSolutionPlan
-->
```

```
<owl:Class rdf:about="IssueSolutionPlan">
  <rdfs:subClassOf rdf:resource="&ExtendedDnS;plan"/>
</owl:Class>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/MailBox -->
```

```
<owl:Class rdf:about="MailBox"/>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/Measure -->
```

```
<owl:Class rdf:about="Measure">
  <rdfs:subClassOf rdf:resource="&ExtendedDnS;task"/>
</owl:Class>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/PEGroup -->
```

```
<owl:Class rdf:about="PEGroup">
  <rdfs:subClassOf rdf:resource="&foaf;Group"/>
</owl:Class>
```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/PEUsergroup -->
```

```
<owl:Class rdf:about="PEUsergroup">
  <rdfs:subClassOf rdf:resource="&ns;Usergroup"/>
```

```

    <rdfs:subClassOf rdf:resource="&foaf;OnlineAccount"/>
  </owl:Class>

  <!-- http://www.ontopolis.net/ontopolis/0.1/PoliticalCandidate
  -->

  <owl:Class rdf:about="PoliticalCandidate">
    <rdfs:subClassOf rdf:resource="PoliticalRole"/>
  </owl:Class>

  <!-- http://www.ontopolis.net/ontopolis/0.1/PoliticalIssue -->

  <owl:Class rdf:about="PoliticalIssue">
    <rdfs:subClassOf rdf:resource="&ExtendedDnS;social-
    description"/>
  </owl:Class>

  <!-- http://www.ontopolis.net/ontopolis/0.1/PoliticalRole -->

  <owl:Class rdf:about="PoliticalRole">
    <rdfs:subClassOf rdf:resource="&ExtendedDnS;agent-driven-
    role"/>
  </owl:Class>

  <!-- http://www.ontopolis.net/ontopolis/0.1/PoliticalSupporter
  -->

  <owl:Class rdf:about="PoliticalSupporter">
    <rdfs:subClassOf rdf:resource="PoliticalRole"/>
  </owl:Class>

  <!-- http://www.ontopolis.net/ontopolis/0.1/SimilarityInfo -->

  <owl:Class rdf:about="SimilarityInfo">
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&DOLCE-Lite;proper-
        part"/>
        <owl:someValuesFrom rdf:resource="&trust;Item"/>
      </owl:Restriction>
    </owl:equivalentClass>
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="similarity"/>
        <owl:someValuesFrom rdf:resource="&xsd;double"/>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>

```

```

<!-- http://www.ontopolis.net/ontopolis/0.1/Support -->
<owl:Class rdf:about="Support">
  <rdfs:subClassOf rdf:resource="&ExtendedDnS;task"/>
</owl:Class>

<!-- http://www.ontopolis.net/ontopolis/0.1/TrustItem -->
<owl:Class rdf:about="TrustItem">
  <rdfs:subClassOf rdf:resource="&trust;Item"/>
</owl:Class>

<!-- http://www.w3.org/2000/01/rdf-schema#Class -->
<owl:Class rdf:about="&rdfs;Class"/>

<!-- http://www.w3.org/2002/07/owl#Thing -->
<owl:Class rdf:about="&owl;Thing"/>

<!-- http://xmlns.com/foaf/0.1/Agent -->
<owl:Class rdf:about="&foaf;Agent"/>

<!-- http://xmlns.com/foaf/0.1/Group -->
<owl:Class rdf:about="&foaf;Group"/>

<!-- http://xmlns.com/foaf/0.1/OnlineAccount -->
<owl:Class rdf:about="&foaf;OnlineAccount"/>

<!--
////////////////////////////////////
//
// Individuals
//
////////////////////////////////////
-->

```

```
<!-- http://www.ontopolis.net/ontopolis/0.1/
      AddPlanPermissionInd -->

<AddPlanPermission rdf:about="AddPlanPermissionInd">
  <rdf:type rdf:resource="owl:Thing"/>
</AddPlanPermission>

<!-- http://www.ontopolis.net/ontopolis/0.1/
      DeletePlanPermissionInd -->

<DeletePlanPermission rdf:about="DeletePlanPermissionInd">
  <rdf:type rdf:resource="owl:Thing"/>
</DeletePlanPermission>
</rdf:RDF>

<!-- Generated by the OWL API (version 2.2.1.1138) http://owlapi.
      sourceforge.net -->
```

B

SIMILARITIES BETWEEN SOLUTION PLANS

PLAN DESCRIPTIONS

| ID | NAME | TAGS |
|----|---|------------------------------------|
| 1 | Finance law reform plan | finance, reform, usury, punishment |
| 2 | Plan of tax reform | taxation, student, reform |
| 3 | Solution of high criminality and drug abuse in Prague | drug abuse, violence, city centre |
| 4 | Complex solution plan of problems with bark beetle | protection, bark beetle, Šumava |
| 5 | Complex strategy for greenery preservation in cities | preservation, city, park |
| 6 | Plan of development of town infrastructure | sidewalk, town, infrastructure |
| 7 | Plan of building-up new school building | building, new school |
| 8 | Safe Prague | jailbird, Prague, harsh sentence |
| 9 | Plan for deforestation of Šumava | bark beetle, Šumava, cutting down |
| 10 | Combating of corruption plan | corruption, bribery |

MUTUAL SIMILARITY VALUES

Following table illustrates the results of the item similarity algorithm described in the section 5.2.5 on page 47 tested on the testing dataset. We emphasized similarities between two distinct plans that exceed currently used threshold 0.3. However, not all of these plans are actually presented to the user as similar, because the similarity is computed only on plans that have joint issue, tag and/or synset. Only plans {4, 9} and {1, 2} are thus presented as similar. Note that more testing especially on real data created by end-users is needed.

| PLAN ID | PLAN ID | SIMILARITY |
|----------|----------|----------------------------|
| 1 | 1 | 1.0 |
| 1 | 2 | 0.46762214219719345 |
| 1 | 3 | 0.23375885277969677 |
| 1 | 4 | 0.0 |
| 1 | 5 | 0.0 |
| 1 | 6 | 0.0 |
| 1 | 7 | 0.0 |

| | | |
|---|----|----------------------------|
| 1 | 8 | 0.0 |
| 1 | 9 | 0.0 |
| 1 | 10 | 0.1609091033435104 |
| 2 | 2 | 1.0 |
| 2 | 3 | 0.23695224492844652 |
| 2 | 4 | 0.0644659904159603 |
| 2 | 5 | 0.0381022082093853 |
| 2 | 6 | 0.04821654738873876 |
| 2 | 7 | 0.0 |
| 2 | 8 | 0.08746058591586224 |
| 2 | 9 | 0.0 |
| 2 | 10 | 0.16567175416663854 |
| 3 | 3 | 1.0 |
| 3 | 4 | 0.024856842733443837 |
| 3 | 5 | 0.11717222849029582 |
| 3 | 6 | 0.1066851642028856 |
| 3 | 7 | 0.0 |
| 3 | 8 | 0.15863501120415022 |
| 3 | 9 | 0.0 |
| 3 | 10 | 0.14366744300380463 |
| 4 | 4 | 1.0 |
| 4 | 5 | 0.058779770084240984 |
| 4 | 6 | 0.12601748369812146 |
| 4 | 7 | 0.0 |
| 4 | 8 | 0.10248638118839866 |
| 4 | 9 | 0.6666666666666666 |
| 4 | 10 | 0.0 |
| 5 | 5 | 1.0 |
| 5 | 6 | 0.48501196765116666 |
| 5 | 7 | 0.0 |
| 5 | 8 | 0.39038607164051564 |
| 5 | 9 | 0.0 |

| | | |
|----|----|---------------------|
| 5 | 10 | 0.0 |
| 6 | 6 | 1.0 |
| 6 | 7 | 0.0 |
| 6 | 8 | 0.23667091115601943 |
| 6 | 9 | 0.0 |
| 6 | 10 | 0.0 |
| 7 | 7 | 1.0 |
| 7 | 8 | 0.0 |
| 7 | 9 | 0.0 |
| 7 | 10 | 0.0 |
| 8 | 8 | 1.0 |
| 8 | 9 | 0.0 |
| 8 | 10 | 0.0 |
| 9 | 9 | 1.0 |
| 9 | 10 | 0.0 |
| 10 | 10 | 1.0 |



SPARQL SIMILARITY QUERIES

POSSIBLE SIMILAR PLANS

Below is a listing of the query for selection of plans possible similar to ?startPlan.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX opol: <http://www.ontopolis.net/ontopolis/0.1/>
PREFIX dolite: <http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#>
PREFIX doledns: <http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#>
PREFIX dolplans: <http://www.loa-cnr.it/ontologies/Plans.owl#>
PREFIX dolmd: <http://www.loa-cnr.it/ontologies/ModalDescriptions.
    owl#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX sioc_c: <http://rdfs.org/sioc/ns#>
PREFIX sioc_t: <http://rdfs.org/sioc/types#>
PREFIX wn: <http://www.w3.org/2006/03/wn/wn20/schema/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sioc_a: <http://rdfs.org/sioc/access#>
```

```
SELECT DISTINCT ?plan (COUNT(?plan) AS ?planCount)
WHERE {
  # plans that shares an issue
  { ?person dolplans:adopts-plan ?plan .
    ?plan dolite:proper-part ?goal .
    ?goal opol:is_solution_of ?issue .
    ?startPlan dolite:proper-part ?g .
    ?g opol:is_solution_of ?issue .
    FILTER (?plan != ?startPlan)
  } UNION {
  # plan that shares a tag
    ?startPlan sioc_c:topic ?tag .
    ?plan sioc_c:topic ?tag .
    ?plan rdf:type opol:IssueSolutionPlan .
    FILTER (?plan != ?startPlan)
  } UNION {
  # plan that shares a synset
    ?s rdf:subject ?startPlan .
    ?s rdf:predicate sioc_c:topic .
    ?s rdf:object ?tag .
    ?s dcterms:subject ?synset .
    ?plan rdf:type opol:IssueSolutionPlan .
    ?s2 rdf:subject ?plan .
    ?s2 rdf:predicate sioc_c:topic .
    ?s2 rdf:object ?tag2 .
    ?s2 dcterms:subject ?synset .
    FILTER (?plan != ?startPlan)
  }
}
GROUP BY ?plan
ORDER BY DESC(?planCount)
LIMIT 5
```

POSSIBLE SIMILAR ISSUES

Below is a listing of the query for selection of issues possible similar to ?startIssue.

```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX opol: <http://www.ontopolis.net/ontopolis/0.1/>
PREFIX dolite: <http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#>
PREFIX doledns: <http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#>
PREFIX dolplans: <http://www.loa-cnr.it/ontologies/Plans.owl#>
PREFIX dolmd: <http://www.loa-cnr.it/ontologies/ModalDescriptions.
    owl#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX sioc_c: <http://rdfs.org/sioc/ns#>
PREFIX sioc_t: <http://rdfs.org/sioc/types#>
PREFIX wn: <http://www.w3.org/2006/03/wn/wn20/schema/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sioc_a: <http://rdfs.org/sioc/access#>
SELECT DISTINCT ?issue (COUNT(?issue) as ?issueCount)
WHERE {
  {
    # issues that share a tag
    ?startIssue sioc_c:topic ?tag .
    ?issue sioc_c:topic ?tag .
    ?issue rdf:type opol:PoliticalIssue .
    FILTER (?issue != ?startIssue) .
  } UNION {
    # issues that share a synset
    ?s rdf:subject ?startIssue .
    ?s rdf:predicate sioc_c:topic .
    ?s rdf:object ?tag .
    ?s dcterms:subject ?synset .
    ?issue rdf:type opol:PoliticalIssue .
    ?s2 rdf:subject ?issue .
    ?s2 rdf:predicate sioc_c:topic .
    ?s2 rdf:object ?tag2 .
    ?s2 dcterms:subject ?synset .
    FILTER (?issue != ?startIssue) .
  }
}
GROUP BY ?issue
ORDER BY DESC(?issueCount)
LIMIT 5

```

SIMILAR USERS

Users similar to the user with the nickname ?startNick are selected by the following query.

```

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX opol: <http://www.ontopolis.net/ontopolis/0.1/>
PREFIX dolite: <http://www.loa-cnr.it/ontologies/DOLCE-Lite.owl#>
PREFIX doledns: <http://www.loa-cnr.it/ontologies/ExtendedDnS.owl#>
PREFIX dolplans: <http://www.loa-cnr.it/ontologies/Plans.owl#>
PREFIX dolmd: <http://www.loa-cnr.it/ontologies/ModalDescriptions.
    owl#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX sioc_c: <http://rdfs.org/sioc/ns#>
PREFIX sioc_t: <http://rdfs.org/sioc/types#>

```

```

PREFIX wn: <http://www.w3.org/2006/03/wn/wn20/schema/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX sioc_a: <http://rdfs.org/sioc/access#>
SELECT DISTINCT ?nick ?fn ?sn (COUNT(?nick) AS ?userCount)
WHERE {
  ?startPerson foaf:holdsAccount/foaf:accountName ?startNick .
  {
    # persons that shares a plan
    ?startPerson dolplans:adopts-plan ?plan .
    ?assocPerson dolplans:adopts-plan ?plan .
    ?assocPerson foaf:firstName ?fn .
    ?assocPerson foaf:surname ?sn .
    ?assocPerson foaf:holdsAccount/foaf:accountName ?nick .
    FILTER(?startPerson != ?assocPerson)
  } UNION {
    # persons that shares an issue
    ?startPerson dolplans:adopts-plan/dolite:proper-part/opol:
      is_solution_of ?issue .
    ?assocPerson dolplans:adopts-plan/dolite:proper-part/opol:
      is_solution_of ?issue .
    ?assocPerson foaf:firstName ?fn .
    ?assocPerson foaf:surname ?sn .
    ?assocPerson foaf:holdsAccount/foaf:accountName ?nick .
    FILTER(?startPerson != ?assocPerson)
  } UNION {
    # persons that shares a group
    ?group foaf:member ?startPerson .
    ?group foaf:member ?assocPerson .
    FILTER(?startPerson != ?assocPerson)
    ?assocPerson foaf:firstName ?fn .
    ?assocPerson foaf:surname ?sn .
    ?assocPerson foaf:holdsAccount/foaf:accountName ?nick .
  }
}
GROUP BY ?nick ?fn ?sn
ORDER BY DESC(?userCount) ?nick

```


COLOPHON

This thesis was typeset with $\text{\LaTeX} 2_{\epsilon}$ using Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL* were used). The listings are typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.) Pictures of graphs and ontologies was created using the **Graphviz** package.

The typographic style is available for \LaTeX via CTAN as "`classicthesis`".

Final Version as of December 9, 2009 at 10:47.